Journal of Visual Languages and Computing (1997) 8, 215–260

Visual Query Systems for Databases: A Survey

Tiziana Catarci*, Maria F. Costabile†, Stefano Levialdi‡ and Carlo Batini*

**Dipartimento di Informatica e Sistemistica, Università di Roma 'La Sapienza', Via Salaria 113, 00198 Roma, Italy, e-mail: [catarci/batini] @infokit.dis.uniroma1.it.; †Dipartimento di Informatica, Università di Bari, Via Orabona 4, 70125 Bari, Italy, e-mail: fcosta@iesi.ba.cnr.it.; and ‡Dipartimento di Scienze dell'Informazione, Università di Roma 'La Sapienza', Via Salaria 113, 00198 Roma, Italy, e-mail: levialdi@dsi.uniroma1.it.*

Received 1 June 1996; revised 4 December 1996; accepted 14 January 1997

Visual query systems (VQSs) are query systems for databases that use visual representations to depict the domain of interest and express related requests. VQSs can be seen as an evolution of query languages adopted into database management systems; they are designed to improve the effectiveness of the human–computer communication. Thus, their most important features are those that determine the nature of the human–computer dialogue. In order to survey and compare existing VQSs used for querying traditional databases, we first introduce a classification based on such features, namely the adopted visual representations and the interaction strategies. We then identify several user types and match the VQS classes against them, in order to understand which kind of system may be suitable for each kind of user. We also report usability experiments which support our claims. Finally, some of the most important open problems in the VQS area are described.
© 1997 Academic Press Limited

1. Introduction

Visual query systems (VQSs) are defined as systems for querying databases that use a visual representation to depict the domain of interest and express related requests. Such systems take advantage of the well-known high bandwidth of the human-vision channel, allowing both recognition and handling of large quantities of information as well as the possibility of using visual feedback to improve the human–computer dialogue. VQSs provide both a language to express the queries in a visual format and a variety of functionalities to facilitate user–system interaction. As such, they are oriented towards a wide spectrum of users, especially novices who have limited computer expertise and generally ignore the inner structure of the accessed database.

In recent years, many VQSs have been proposed in the literature (see e.g. Catarci and Costabile [93, 94], Catarci *et al.* [95], Cooper [110], Kennedy and Barclay [126] and Sawyer [146]). They adopt a range of different visual representations and interaction strategies (a very preliminary survey is by Kim [127]). The huge number of proposals testify the growing interest in the field. However, one can notice a certain confusion in identifying VQSs within the broader class of generic visual systems. In particular, it seems that any system exhibiting a visual-flavored interface and dealing with information repositories falls into the same generic class, while the purpose of the system is a first discriminatory aspect. For instance, toolkits for interface design deeply differ from VQSs, and information retrieval is not the same as database querying. Secondly, systems dealing with image data, non-structured text data, geographical data or alphanumeric data have dissimilar characteristics, i.e. the kind of data influences the kind of system.

In this paper, we first identify precisely the VQSs we are interested in. They are the VQSs used for querying traditional databases, and, as such, they deal with alphanumeric data only. This is still the most common type of data. It is our opinion that our analysis can be the basis for further studies dealing with more advanced types of data, namely maps and images, but also video.

The main goal of this paper is to survey and classify the VQSs for traditional databases. In choosing the classification dimensions, we have taken into account the 'user's point of view'. It follows that we are primarily interested in the interactive aspects of the user–system dialogue. Such a dialogue is mainly influenced by both the visual representations displayed on the screen and by the interaction strategies offered to the user. Therefore, visual representation and interaction strategy are our two classification criteria.

The paper is organized as follows: Section 2 provides the VQS background. Section 3 describes the classification criteria. In Sections 4 and 5, VQSs are surveyed under the light of such a classification. In Section 6, we identify several user types and match the VQSs against them in order to understand which kind of system may be suitable for each type of user. We also report usability experiments which support our claims. In Section 7, we discuss some formal issues and mention the VQSs that have addressed them. Finally, Section 8 highlights significant, yet still unsolved, problems which could improve present VQSs.

2. Background

One of the most compelling mandates for any kind of interactive system is aiming the interaction at the user who will work with the system, rather than at the system itself (by 'user' we refer in this paper to the 'end-user', i.e. the person working with the VQS). The system should be tailored to help the users perform the tasks they have in mind. Therefore, the characteristics of the classes of users that will be working with a particular interface and the tasks such users need to perform should be well understood.

The purpose of the systems we are going to survey in this paper is to provide access to the information contained in a database. The main users' tasks are understanding the database content, focusing on meaningful items, finding query patterns and reasoning on the query result. These tasks require specific techniques to be effectively accomplished, and such techniques involve typical spatial activities such as pointing, browsing, filtering and zooming. Consequently, we can say that a visual user interface is the most adequate for the above tasks, at least from the user's point of view, which is our working viewpoint (see also Rohr [143]). In the rest of the paper, we will not address system-oriented questions, such as what the system data model is, how archives are organized, etc. Indeed, these aspects are uninteresting from the user's perspective.

It is worth noting that most people interacting with computers see only the system interface. Thus, it becomes a very important component of a software system from the design phase onwards. This is in contrast with the old-fashioned developmental approach, where the interface was considered only from an aesthetic point of view and therefore was added only at the last stage of the system development.

2.1. Database Querying

In this work we are interested in describing query interfaces for general purpose database systems that contain alphanumeric data only. A database can be described in terms of a *data model*, i.e. a set of concept types for organizing the reality of interest, so that it can be processed by a Data Base Management System. Existing data models are equipped with suitable concept types in order to represent information such as:

1. *objects* of the real world, together with their meaningful properties (e.g. Charles's age is 40);
2. *relationships* among objects (e.g. Charles is married to Ann);
3. *classes of objects* (e.g. the class of 'people');
4. *relationships among classes* of objects (e.g. people live in towns).

Objects and their relationships constitute the so-called *extension* of the database, while classes of objects and relationships among classes form the so-called *intension* of the database (also called database *schema*).

While databases are designed, created and modified by professionals, many different people access databases specifically to extract information. To help achieve this aim, special purpose languages, called *query languages*, have been defined. A query language is a set of formally defined operators which allow requests to be expressed to a database, when suitably composed. The execution of the query produces results which are extracted from the stored data and are coherent with the intended meaning of the request. Until now, the most widely used database query languages have actually been programming languages [112] which require knowledge about language syntax, technical background and information of both the system application domain and its interaction mechanisms. Such languages do not help to understand the meaning of data, nor do they provide any guidance in satisfying the user's needs. In general, they do not fulfill the requirements of user friendliness and ease of use.

Recently, the growth of database users, both expert and novice, has led to the development of a number of interfaces based on different principles whose main purpose is to facilitate the human–computer interaction. Two different main approaches have been suggested:

1. using a restricted natural language;
2. using *direct manipulation* languages [149], characterized by both the visibility of objects of interest and the replacement of a command language syntax by the direct manipulation of objects.

The first approach is very difficult to apply within a general purpose database system for an unrestricted domain, and is therefore outside the scope of this paper.

The second approach is based on an interaction style widely used in existing commercial interfaces. In fact, the availability of low-cost graphical devices has given rise in recent years to a widespread diffusion of products that have gained popularity due to their increased visual content. A rapid visual feedback, another feature of direct manipulation interfaces, provides evaluative information for every executed user action. Several authors singled out a number of usability features that can be obtained with direct manipulation visual techniques: (1) shortening of the distance between the user's mental model of reality and the representation of such reality proposed by the computer; (2) reduction of the dependency on the native language of the user; (3) ease in learning of the basic functionality of the interaction; (4) high efficiency rate obtained also by expert users, partly because of the possibility of defining new functions and features; and (5) significant reduction in the error rate [157].

Direct manipulation is successfully used in conjunction with windows, icons, menus and pointers in the so-called WIMP (Window, Icon, Menu, Pointer) interfaces [115], where metaphors are also adopted to increase the initial familiarity between user and computer applications [96].

Once WIMP interfaces are designed for querying databases, the query language operators have to be rendered in terms of appropriate direct manipulation sequences. Such sequences cannot be applied on the formal data model concepts, but on a proper external *representation* of such concepts which must be easily perceived by the user (see also Haber *et al.* [121]). As such, the representation has to be close to the reality the user lives in. When the chosen representation is visual, we can say that we are in the presence of a *visual query language* (VQL). VQLs are the languages implemented by VQSs. VQLs can also be seen as a particular subclass (aimed at extracting information from databases) of the more general class of *visual languages*, as introduced by Chang [103]. More precisely, visual query languages are a subclass of *visual programming languages*, i.e. languages whose constructs are visual, but which handle objects that do not necessarily have an intrinsic visual representation (e.g. alphanumeric data in traditional databases). Note that another language class singled out by Chang, namely *visual information processing languages*, is far from our purposes, since these are conventional languages enhanced with subroutine libraries to handle objects that are intrinsically visual, such as images.

The next subsection is devoted to expanding on the kinds of visual formalisms we are interested in.

2.2. Visual Formalisms

A successful sign system conceived by humans for the purpose of storing, communicating and perceiving essential information is based on bi-dimensional visual signs such as diagrams, pictures, photographs and geographic maps. Visual signs are characterized by a high number of sensory variables: size, intensity, texture, shape, orientation and color [89]. For instance, Tufte suggests many different purposes for which color may be employed: '...to label (color as noun), to measure (color as quantity), to represent or imitate reality (color as representation) and to enliven or decorate (color as beauty)' [152]. By exploiting the multi-dimensionality of visual representations, people are allowed to perform, in a single instant, a visual selection. Other constructions, for example a linear text, do not permit this immediate grasping, since the entire set of correspondences may be reconstructed only in the user's memory.

The main purpose of adopting a visual representation in a query system is to communicate clearly to the user the information content of the database(s), concentrating on essential features and omitting unnecessary details. Such information is internally structured in several manners that mainly depend on the data model characteristics, but it must be rendered at the interface level in such a way that any user can easily grasp it. However, the interface visual representation has to be mapped in terms of internal database concepts in order to be dealt with by the system. To come up with this dual nature, a visual representation has to be based on a so-called *visual formalism*, a term introduced by David Harel as follows [122]: 'The intricate nature of a variety of computer-related systems and situations can, and in our opinion should, be represented via *visual formalisms*; visual because they are to be generated, comprehended and communicated by humans; and formal, because they are to be manipulated, maintained and analyzed by computers'. Visual formalisms include familiar objects such as tables, diagrams, icons, etc. They are the components of the visual representations as proposed by the existing VQSs (see Section 4).

It is clear that the importance of representation friendliness decreases according to the user's experience, since the user's technical background allows him/her to learn new abstract constructs with a reasonable effort. Whereas, when the user is a novice, comfortable representations are highly desirable, since he or she prefers to interact with something similar to the reality lived in, without being acquainted with the existence of the underlying abstract model.

In the next sections we will see that the kind of visual representation adopted is the most distinguishing aspect for classifying the VQSs since it influences all the subsequent choices in the human–computer dialogue.

3. Classification Criteria for VQSs

Many VQSs have been recently proposed in the literature. In this section we introduce the criteria for a VQS taxonomy. A first attempt to classify query languages on the basis of their *ease-of-use* is by Reisner [140]. A more recent query language taxonomy has been proposed by Jarke and Vass [124]. In their analysis, Jarke and Vassiliou include many kinds of query languages, namely traditional and new generation ones, where the use of more senses for the interaction is the main distinguishing factor between the two classes. All query languages are evaluated on the basis of two dimensions: functional capabilities and usability. The former is related to both the language power (how much a user can do with the language) and the alternatives the user has for output presentation. The latter is related to the query formulation effort, i.e. the user's effort to work with the system.

In our analysis, we concentrate on VQSs. In order to classify these systems, the first criterion we consider is the *visual representation* adopted to present the reality of interest, the applicable language operators and the query result. The query representation is generally dependent on the database representation, since the way in which the query operands (i.e. data in the database) are presented constrains the query representation. Therefore, we will consider a unique classification schema for both database and query representations. Based on such a schema, systems are organized into four classes depending on the adopted visual formalism, i.e. form, diagram, icon or a combination of them, as will be described in Section 4.1. On the other hand, the visual representation used to display the query result can be different from the database representation, giving rise to a different classification (see Section 4.2). This is mainly due to the fact that very often what is visualized for the query purpose is the schema of the database, while the actual database instances constitute the query result to be displayed to the user.

The goal of the people working with a VQS is to retrieve the desired data. This is usually accomplished through the following two main activities:

1. *Understanding the reality of interest.* The goal of this activity is the precise definition of the fragment of schema involved in the query. Generally, the schema is much richer than the subset of concepts which are involved in the query. The first step corresponds to the identification of concepts that are useful for the query, and the result is a query subschema, i.e. the static representation of such concepts.
2. *Formulating the query.* The query subschema can be manipulated in several ways, according to the available query operators. The goal of query formulation is to formally express the operands involved in the query, with their related operators.

The second criterion for the proposed VQS classification refers to the *interaction strategies* provided to perform the above two activities. They will be illustrated in Section 5.

4. Visual Representations

Many authors agree that visual representations are effective for expressing different kinds of knowledge. One of the main advantages relies on their capacity to allow perceptual inferences, replacing arduous cognitive comparisons and computations. Of the taxonomies of visual representations that have been proposed, some simply rely on the authors' intuitions [89, 139, 153], while a recent paper discusses empirical work carried out to discover and elaborate the basis on which people organize visual information [130]. The result of this work is a classification of visual representations into 11 categories, ranging from tables to maps, icons, pictures and diagrams of different kinds. From the analysis of the existing VQSs, we notice that various kinds of tables, diagrams and icons are the only adopted visual representations. Obviously, this is true for the VQSs we are interested in, i.e. those used for accessing alphanumeric data, while VQSs for geographical or multimedia systems largely adopt maps and pictures.

In this section we describe and exemplify the different VQS representations. Pioneering systems [101, 125, 133, 134, 144, 147, 155, 156] adopted only visual data representations, and left a textual format for the query. They therefore cannot be treated as VQSs, whose mandatory feature is the use of visual mechanisms for query formulation.

More recent systems also offer the possibility of visualizing the query result. In Section 4.1, VQSs are classified according to the database and query representation, while in Section 4.2 they are classified according to the query result representation.

4.1. Database and Query Visual Representations

Table 1 shows the VQS classification according to the visual representation used for database and query (numbers indicate the VQS bibliographical references). Note that here and in the remaining of this paper, every classification table includes the VQSs which match the features described in the table only.

In the following, each one of these representations is illustrated, together with a brief discussion on the main features of the visual formalism it is based on. Note that we use the more general term *form* instead of table.

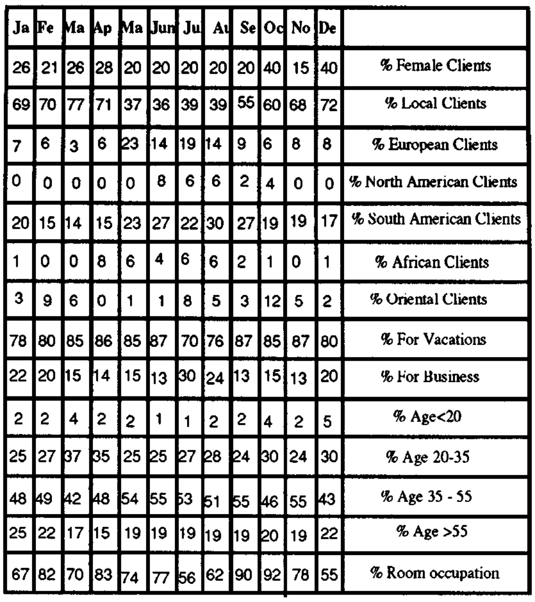
Table 1. VQS classification according to the database and query representation

Representation	Systems
Form-based	28, 29, 32, 33, 38, 39, 41, 42, 53, 61, 64, 69, 71, 75, 76, 79, 80
Diagram-based	2, 3, 4, 5, 7, 8, 9, 10, 12, 13, 16, 17, 18, 19, 20, 21, 22, 24, 27, 30, 31, 34, 35, 37, 40, 43, 44, 45, 47, 51, 52, 55, 56, 57, 59, 60, 62, 63, 65, 66, 72, 77, 78
Icon-based	14, 54, 75
Hybrid	
Form and Diagram	1, 11, 25, 26, 46, 48, 49, 50, 58, 68, 70, 74
Diagram and Icon	6, 36, 67
Form, Diagram and Icon	15, 23

4.1.1. Form-based

A form is a named collection of objects having the same structure. It is the first attempt to leave the textual mono-dimensional space, exploiting the bi-dimensionality of the screen. It facilitates non-expert users by capitalizing on the natural tendency of people to use regular structures and/or to organize data into tables. The main characteristic of a computer form is that it is a structured representation which corresponds to an abstraction of conventional paper forms. A form can be seen as a rectangular grid having components which may be any combination of cells and groups of cells (or subform). A form cell is the smallest unit of data that can be referenced in a user's application [150]. A form is a generalization of a table, in the sense that table components are usually elementary cells and no nesting is allowed. The relationships represented through a form can be among cells, subsets or the overall set, thus providing the user with three information levels [89]. This allows questions regarding single cells (elementary question), cell subsets (intermediate question) or the overall set of cells (overall question) to be answered. In Figure 1, a form representing the hotel occupancy in one year is shown. An elementary question could be: 'What is the percentage of clients older than 55 in May?'. The answer to such a question may be easily seen from a single correspondence in the form. An intermediate question could be: 'What is the percentage of female clients in winter?'. The answer comes from observing a group of correspondences in the form. Finally, an overall question could be: 'What was the trend of the room occupation during the entire year?'.

Initially, forms were used in the framework of the relational model [108], where prototypical tables are visualized and queries are expressed by filling table fields with suitable operands. In QBE [80], the first system adopting a form-based representation, only the intensional part of relations is shown, while the extensional part is filled by the user in order to provide an example of the requested result. In more recent form-based representations (e.g. Houben and Paredaens [41], Shirota *et al.* [69] and Wegner [76]) both the intensional and the extensional part of the database may be manipulated by the user. Such systems allow visualization of complex forms, together with zooming mechanisms in order to focus on different parts of the database schema.



Ja	Fe	Ma	Ap	Ma	Jun	Ju	Au	Se	Oc	No	De	
26	21	26	28	20	20	20	20	20	40	15	40	% Female Clients
69	70	77	71	37	36	39	39	55	60	68	72	% Local Clients
7	6	3	6	23	14	19	14	9	6	8	8	% European Clients
0	0	0	0	0	8	6	6	2	4	0	0	% North American Clients
20	15	14	15	23	27	22	30	27	19	19	17	% South American Clients
1	0	0	8	6	4	6	6	2	1	0	1	% African Clients
3	9	6	0	1	1	8	5	3	12	5	2	% Oriental Clients
78	80	85	86	85	87	70	76	87	85	87	80	% For Vacations
22	20	15	14	15	13	30	24	13	15	13	20	% For Business
2	2	4	2	2	1	1	2	2	4	2	5	% Age<20
25	27	37	35	25	25	27	28	24	30	24	30	% Age 20-35
48	49	42	48	54	55	53	51	55	46	55	43	% Age 35 - 55
25	22	17	15	19	19	19	19	19	20	19	22	% Age >55
67	82	70	83	74	77	56	62	90	92	78	55	% Room occupation

Figure 1. Information on the hotel clients during a year (redrawn from Bertin [89])

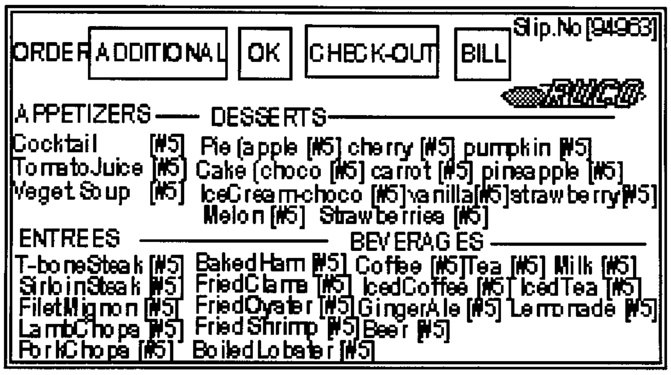
An interesting approach suggested in EMBS [69] aims at overcoming the inherent limitation of forms in visualizing associations among data. EMBS displays two typical elements: cells and buttons. Cells contain data values (organized in a table) which may be specified by the user by supplying example values (see Figure 2). Selecting a button or changing a cell value triggers a data manipulation. Six primitive functions are provided, which may be further combined to perform complex actions.

TableTalk [33] employs a form-based representation for a wide family of models, including entity-relationship (E-R in the sequel) [107], network, functional and more typical object-oriented models. TableTalk is based on a functional query language, and it uses a table-heading metaphor to express the semantics of the language and spatial arrangements of visual elements to express functional combinators.

4.1.2. Diagram-based

The data contained in a form may be better understood if we use some kind of graphical representation that can show relationships better among those data. For instance, rows of numbers can be transformed into a profile in which the height of each column is proportional to the value of the corresponding number (see Figure 3 from Bertin [89], concerning meat production in European countries). Furthermore, those row profiles can be arranged in different ways for highlighting other relationships amongst the data.

The right part of Figure 3 fits Bertin's definition of a diagram well: 'The graphic is a diagram when the correspondences on the plane can be established among all the elements of one component and all the elements of *another* component' [89]. We use the word diagram with a broader meaning, referring to any graphics that encode



Slip.No [94963]

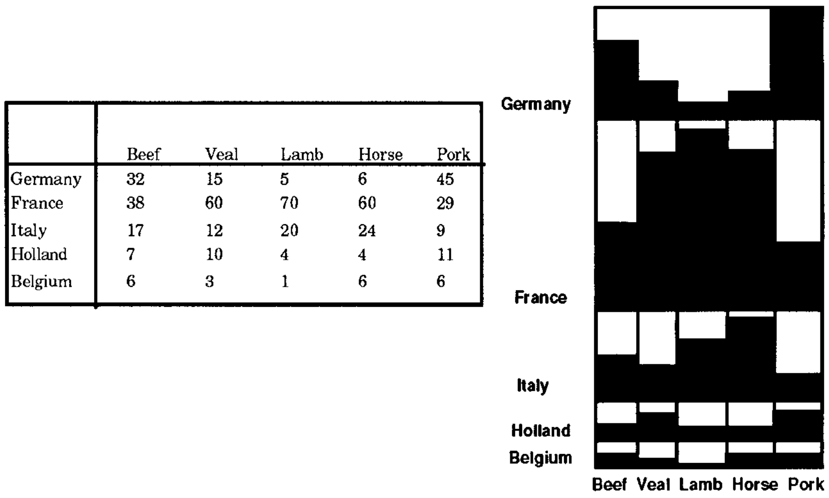
APPETIZERS —— DESSERTS

Cocktail [#5] Pie (apple [#5] cherry [#5] pumpkin [#5]
TomatoJuice [#5] Cake (choco [#5] carrot [#5] pineapple [#5]
Veget.Soup [#5] IceCream-choco [#5] vanilla [#5] strawberry [#5]
Melon [#5] Strawberries [#5]

ENTREES —— BEVERAGES

T-boneSteak [#5] BakedHam [#5] Coffee [#5] Tea [#5] Milk [#5]
SirloinSteak [#5] FriedClams [#5] IcedCoffee [#5] IcedTea [#5]
FiletMignon [#5] FriedOyster [#5] GingerAle [#5] Lemonade [#5]
LambChops [#5] FriedShrimp [#5] Beer [#5]
PorkChops [#5] BoiledLobster [#5]

Figure 2. Example of form-based representation in EMBS (from Shirota *et al.* [69])



	Beef	Veal	Lamb	Horse	Pork
Germany	32	15	5	6	45
France	38	60	70	60	29
Italy	17	12	20	24	9
Holland	7	10	4	4	11
Belgium	6	3	1	6	6

Figure 3. Form and diagram representing the meat production in some European countries (from Bertin [89])

information using position and magnitude of geometrical objects and/or show the relationships among components. This definition includes the notions of graphs, network charts, structure diagrams and process diagrams as described by Lohse *et al.* [130].

Diagrams frequently adopted in VQSs use as basic (visual) components either points or simple geometrical figures (squares, rectangles, circles, etc.). Generally, a diagram uses visual components that have a one-to-one correspondence with specific concept types. Lines denote logical relationship types among elements. Sometimes labels are also included in diagrams for denotational purposes [see in Figure 4(a) an example of a diagram where rectangles represent entities in the E-R model and diamonds represent relationships]. In a diagram, if we modify its layout by following certain rules, its content can show new relationships [117]. For instance, by means of spatial proximity, the layout may easily convey the notion of similarity and the presence of a kernel concept around

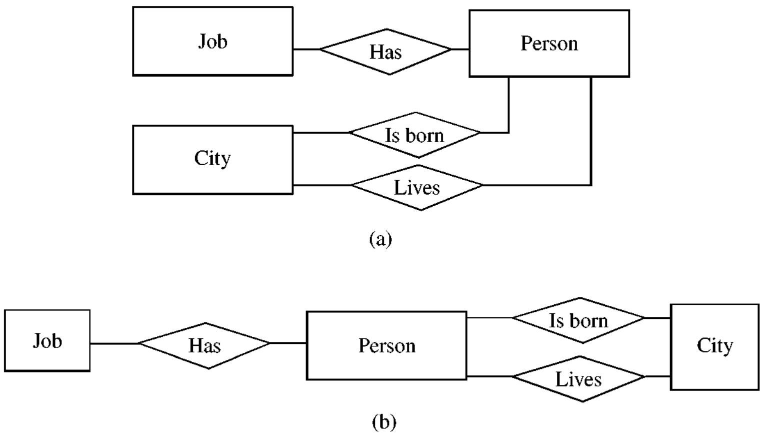


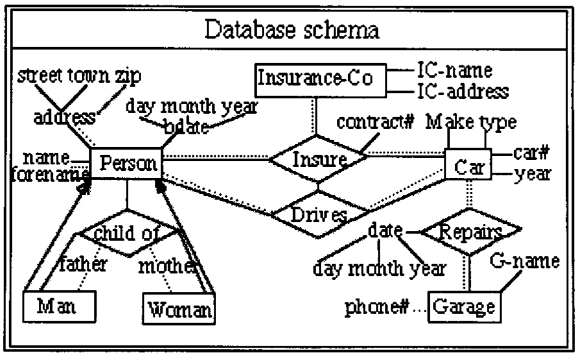
Figure 4. (a) Example of an E-R diagram. (b) Person is highlighted as the most important concept

which all other concepts gravitate. The diagram of Figure 4(a) may be redrawn as Figure 4(b), thereby stressing the role of the entity Person, placed in the central part of the diagram.

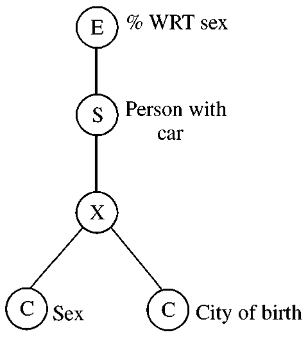
While most relationships are represented by means of connections, special relationships, namely structural relationships, are conveniently represented via inclusions of geometrical figures, such as in the Eulero–Venn diagrams and in Harel hygraphs [122].

Diagrams emerge as the most popular visual formalism used in existing VQSs. Examples of diagrams can be seen in Figure 5. Figure 5(a) refers to SUPER [27], which visualizes the database schema by using an extension of the well-known E-R diagrams. Rectangles are used for entities, diamonds for relationships, arrows for the IS-A special relationships and labels, linked to the owner entity or relationship, for attributes. Other representations use a single symbol for all concept types, where each type is identified by a label or a name [see Figure 5(b), showing an example of diagrams used in GRASP [12]]. Figure 5(c) shows a diagram in PICASSO [45], where labels denote attributes and closed lines represent relations among attributes. This representation is particularly suited to the relational model, since intersection areas bordered by closed lines effectively show common attributes among relations which correspond to logical links. Structured symbols are adopted in some interfaces, as in ISIS [35] [see Figure 5(d)], where inner rectangles represent component objects and outer rectangles denote complex objects.

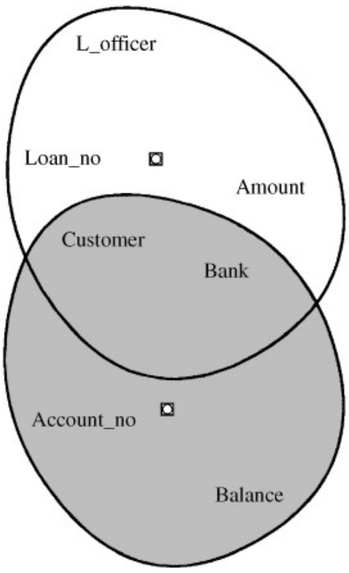
All the above systems use 2D representations, while some VQSs adopting 3D diagrams have recently been proposed [8, 66]. These systems are based on the idea that the introduction of a third dimension can increase the representational power, as shown in the precursor work on general interactive 3D environments developed at Xerox [142]. As an example of VQS using 3D diagrams for both query formulation and result presentation (see next subsection), WINONA shows a variety of 3D visual representations of an object-oriented database schema [66]. In the current implementation, a detailed view of the information about an object can be obtained by double clicking on an object shown in a 3D visualization. This causes a pop-up window to appear, displaying the object attributes and methods in the form of scrollable lists. AMAZE



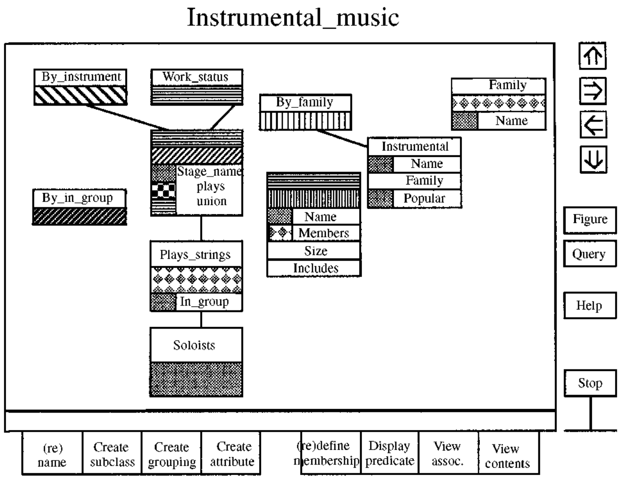
(a)



(b)



(c)



(d)

Figure 5. Examples of diagrammatic representations

employs a 3D display in order to represent a complex schema with a large number of entity classes and relationships, thus trying to minimize the 'wires and meshes' problem, i.e. the number of intersecting lines and boxes [8]. Color is also used as a further encoding technique to distinguish among different roles of entities during an interaction session. A recent proposal examines the possibility of using two interesting 3D visualization metaphors, namely the *search tree* and the *relevance sphere*, as basic components of the interface to an object oriented database system [123].

Concerning aspects related to the query, diagram-based (or diagrammatic) representations adopt, as typical query operators, the selection of visual elements, the traversal on adjacent elements and the creation of a bridge between disconnected elements. In some systems, like GORDAS [31] and SUPER [27], the navigation path is implicitly expressed by transforming the query schema into a tree where nodes appear in the selected order.

Finally, some recent proposals allow the users to define their own visualization, thus creating tailored displays of data. In DOODLE [19] the user-defined visualization can be used both in querying the database and in defining new visualizations.

4.1.3. Icon-based

In computer science, an icon can be defined as a segmented, stylized image. Image segmentation implies the extraction of a single component from the background while stylization refers to a representation made by a small number of significant lines that all together constitute the icon. See, for instance, the house icon on the left of Figure 6, in which only simple geometrical shapes are used.

VQSs need to represent not only images of real objects, but also abstract concepts, actions or processes. If we want to represent a computer process, we are forced to extend the icon scope, since we do not have a natural visual counterpart and cannot exploit the pictorial similarity between such icon and the addressed abstract concept. New icons must be designed by exploiting different correlation modalities such as analogy, methonimics, convention, etc., which, in principle, are more typical for symbols (for a semiotic definition of symbol, see Eco [117]). We see some examples in Figure 6, where, going from left to right, the second icon is designed to indicate a restaurant by analogy, the third icon radioactivity by convention and the last one a gasoline station by methonimics. In summary, we can agree with the following definition of an icon in the context of computer science: 'the icon is a visually segmented object which tells the viewer about an inside message or information (concept, function, state, mode, etc.) assigned by the designer' [118].

The aim of an icon is to represent (or remind of) a certain concept, while diagrams favor the visualization of relationships between concepts (denoted by means of labels). Moreover, in diagrams the description of relationships does not use symbols but rather links between symbols, and the description of links is provided without ambiguity [89]. The icon has a significant metaphorical power, i.e. it can be interpreted as a visual metaphor, as when a red cross on a white background refers to the availability of medical care. In contrast, the only metaphorical power of the diagram comes from its attached labels [120].

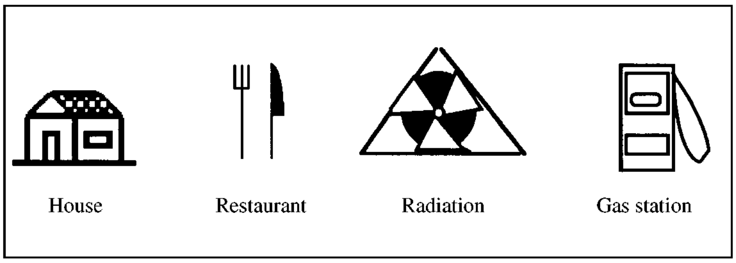


Figure 6. Examples of icons

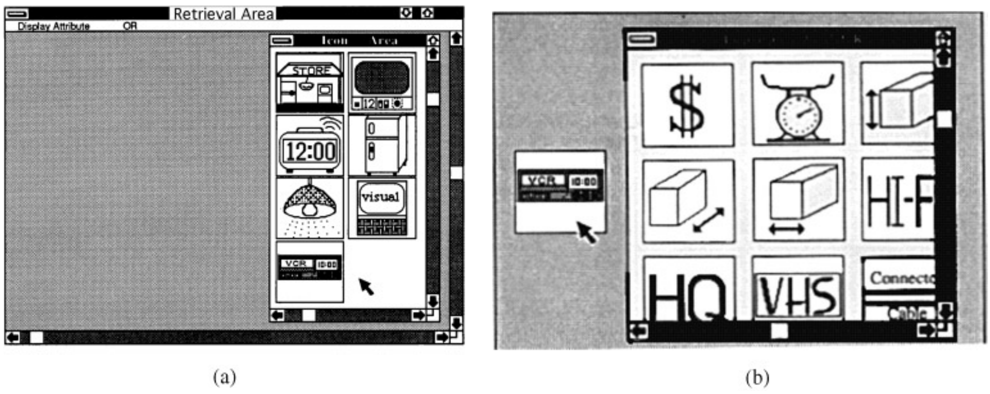


Figure 7. Example of iconic representation (from Tsuda *et al.* [73])

The icon-based (or iconic) VQSs use sets of icons which denote both the entities of the real world and the available functions of the system. A query is expressed primarily by combining icons according to some spatial syntax. The icon-based systems represent an improvement in the friendliness of the human–computer interaction. In fact, the icons allow users to easily grasp the scenario for their manipulations. Generally, in these systems the database schema is not shown. In fact, these VQSs are mainly addressed to users who are not familiar with the concepts of data models and may find it difficult to interpret even an E-R diagram. In Figure 7, an example taken from IconicBrowser [73] is shown, where icons represent objects of an electrical appliance database. The initial display of IconicBrowser is shown in Figure 7(a). By clicking on the icon representing a class of objects, for example video recorders (VCR), all the icons representing operations applicable to the VCR objects are shown in a pop-up menu [see Figure 7(b)].

When designing iconic systems, a crucial problem is how to construct icons that can express a meaning which is objective to humans; quite often the same image conveys different meanings to different people. It would be desirable to have some criteria for constructing icons which could carry an unambiguous meaning as much as possible. Many authors have approached this problem [88, 118, 119]. In Liu and Tai [129] a set of icon construction principles is given in order to eliminate the ambiguity due to the difference between the icon meaning and the mental meaning evoked in the observer. At the present time, no standard can yet be imposed to the set of icons used in different applications; as a consequence, when the number of icons increases, the discrimination power tends to decrease in the interpretation of the icons. Overcrowding of icons can be controlled by using composition mechanisms (syntax rules). In Chang [103] some composition rules are described and icon operators are introduced, perhaps changing either the pictorial part (the actual icon drawing) or the semantic part (the icon meaning) of an icon, thus creating a new icon.

4.1.4. *Hybrid*

The hybrid representation uses an arbitrary combination of the above three visual formalisms, either offering the user various alternative representations of databases and queries, or combining different visual formalisms into a single representation. From an

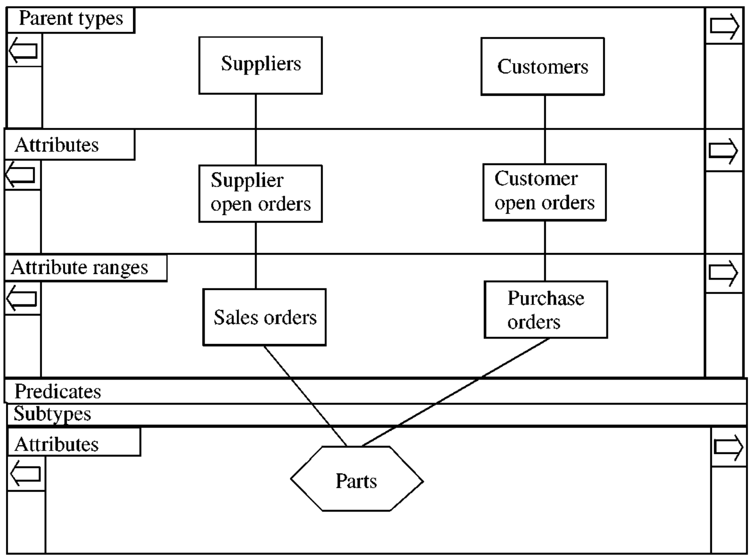


Figure 8. Hybrid approach in SKI [Reproduced with permission from M. P. Consens, I. F. Cruz and A. O. Mendelzon. Visualizing queries and querying visualizations. *ACM Sigmod Record, Special Issue on Advanced User Interfaces* 21. © 1992 The Association for Computing Machinery]

analysis of VQSs [86, 87], it emerges that most systems adopt more than one visual formalism, but often one of them is predominant. In this case, the system can be classified as adopting the representation paradigm based on that specific formalism. However, in the so-called hybrid VQSs, the different visual formalisms share the same significance. In the following, we highlight the various types of hybrid representations adopted in VQSs so far, i.e. representations using: (1) forms and diagrams; (2) diagrams and icons and (3) forms, diagrams and icons.

In the first case, diagrams are generally used to describe the database schema. Forms are primarily employed either to show more detailed information, as for example properties of an entity, its instances, etc., or to formulate the queries. A particular use of forms and diagrams is proposed in SKI [46]. As shown in Figure 8, the screen is divided into a variable number of horizontal stripes. For instance, the top stripe represents parent types, which may be any one of the types or subtypes in the schema. The next stripe represents attributes of the parent types that can be shown, changed, etc. The user may scroll up or down to view all stripes. Any stripe may be collapsed from the screen (this has been done for Predicates and Subtypes stripes in Figure 8). The graph shown in Figure 8 illustrates how the two types *suppliers* and *customers* (previously selected by the user) are related. The square at the end of each stripe indicates that certain tokens have ‘fallen off’ the stripe. By using an arrow, the user may scroll to see the end of the stripe and find the tokens.

In VQSs using diagrams and icons, diagrams again describe the database schema, while icons are used either to represent specific prototypical objects (e.g. a generic instance of a person) or to indicate actions to be performed. For instance, SICON [36] represents a schema using both diagrams and icons. Indeed, the user may visualize the schema in the diagrammatic representation, select an entity type and express a complex query or may simply formulate the query by selecting the appropriate icons. In both cases the result of the query is shown with icons (see next subsection).

X-VIQU [15] is an example of a query system employing three different visual formalisms, namely forms, diagrams and icons. The tree-like structures are interactively constructed while the user formulates a query. The subject initially chosen is shown at the root. At each successive step a new level is added to the tree, corresponding to the choice of a more specific query. An appropriate symbol is placed beside the name of the node if it represents information explicitly stored in the database. More information about a node in the tree is visualized by means of a form; a menu of icons is provided allowing specific actions to be performed.

4.2. Query Result Visual Representations

Similar to most graphical user interfaces, the VQSs that have been developed so far have stressed mainly the user input aspects of the interaction and have given little thought to the output data visualization. The results are usually presented by means of structured text, without considering other possible display formats. Conversely, an appropriate visualization of the query result allows the user to better capture the relationships amongst the output data.

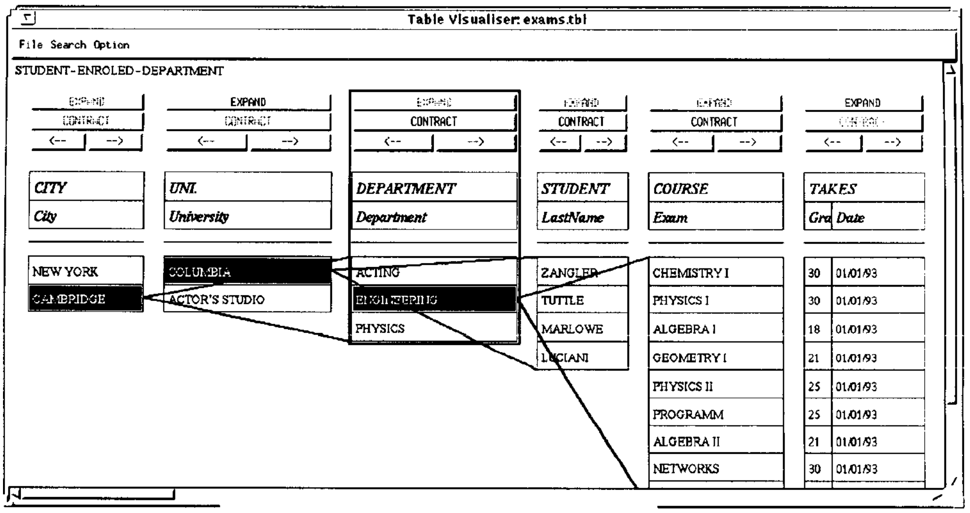
The VQS classification on the basis of the representation primarily used for visualizing the query results is shown in Table 2.

Under *form-based* we enclose all systems that show the result organized in tables, or in more general lists that can easily be scrolled. This class is the most populated, since the table is the typical representation of the relational model. As an example, QBD* [4] presents the query results as relational tables in a compact form, by visualizing only distinct values [145]. Useless and sometimes confusing duplication of values are avoided (see Figure 9), and the user can request to visualize all attribute values at any moment, by a simple click on the mouse.

With reference to *diagram-based* techniques, as we have already explained in Section 4.1.2, we use the word diagram with a broad meaning, referring to any graphic that encodes information by using position and magnitude of geometrical objects. Several kinds of diagrams are employed by the different VQSs, also to visualize the query result. Some VQSs (see e.g. Catarci and Tarantino [13] and Consens and Mendelzon [18]) show the query result through the same kind of graph used for formulating the query, by filling graph nodes with appropriate values. AMAZE is one of the few VQSs employing 3D graphs [8]. The data are shown as a 3D snapshot of the *n*-dimensional results. Different methods of result visualization are also planned to be made available to the user. One of

Table 2. VQS classification according to the query result representation

Representation	Systems
Form-based	4, 21, 23, 29, 33, 39, 44, 45, 50, 62, 66, 79
Diagram-based	1, 8, 13, 18, 40, 53, 56, 70
Icon-based	36, 73
Hybrid	
Form and diagram	11, 27



STUDENT-ENROLED-DEPARTMENT

CITY	*UNI.*	*DEPARTMENT*	*STUDENT*	*COURSE*	*TAKES*	
City	*University*	*Department*	*LastName*	*Exam*	*Gra*	*Date*
NEW YORK	COLUMBIA	ACTING	ZANGLER	CHEMISTRY I	30	01/01/93
CAMBRIDGE	ACTOR'S STUDIO	ENGINEERING	TUTTLE	PHYSICS I	30	01/01/93
		PHYSICS	MARLOWE	ALGEBRA I	18	01/01/93
			LUCIANI	GEOMETRY I	21	01/01/93
				PHYSICS II	25	01/01/93
				PROGRAMM	25	01/01/93
				ALGEBRA II	21	01/01/93
				NETWORKS	30	01/01/93

Figure 9. Result representation in QBD*

these employs tree structures, based on the cone-tree approach [142]. Powerful result visualizations are those provided by Ahlberg and Shneiderman [1] and Shneidermann *et al.* [70]. The work presented by Shneidermann *et al.* [70] refers to a real-estate database about houses in the Washington D.C. area. Retrieved houses are indicated by bright points on the Washington map which is displayed on the screen. The system described by Ahlberg and Shneidermann [1] is called FilmFinder, and visualizes information about movies by means of *starfield displays*, which show database objects as small selectable spots (either points or 2D figures). The displayed data can be filtered by changing the range of values on both the Cartesian axes. The query result fits on a single screen and the system recomputes quickly, i.e. within a second, the new data display in response to the user's requests. This property, called near real-time interactivity, ensures high usability [136].

Icon-based representations use icons to visualize the query result. An example is SICON [36] which shows a set of occurrences of an entity by means of copies of the same icon representing that entity; an identification label (attached to each copy of the icon) is used to distinguish the occurrences. This is very effective if we want to quickly know the kind of the data retrieved. Moreover, when the user finds an entity occurrence of special interest, he or she can obtain the connected relationship occurrences represented in terms of icons.

As we said in Section 4.1, the *hybrid* representation offers to the user various alternative visualizations. In SUPER [27], a browser that provides two data display modes is implemented. The result of a query is returned to such a browser in order to be presented to the user. In the form-based mode, occurrences of the object or relationship types are displayed through form-like representations; in the graphical mode, E-R-like diagrams show the currently examined occurrences which can then be directly manipulated by the users.

An effective system should adapt the presentation to the current task, domain and user; in order to accomplish this, it should use different presentation modalities and contain knowledge of the above three elements (task, domain, user). In this way, the information will be presented in the best modality to be fully perceived and understood. MIMESIS, which actually proposes a multimodal interface using both a visual interaction style and a natural language style, is one of the first VQSs implementing a graphical presentation module that chooses the graphics which better express the data to be visualized [53]. Such a module also allows the graphs displayed in the output window to be manipulated by using graphical operators. Since MIMESIS is the interface to a system which collects and visualizes data on air pollution, the output graphics consist of thematic maps of cities, graphs such as bar charts, XY plots, area plots, multi bar charts, pie charts and also map animation. The idea of the graphical presentation module originated from a pioneering system for information display which automatically draws graphical presentations of information [131].

A new proposal is to present the query result with a simulation of a real environment (i.e. a virtual one) that depicts a situation familiar to the user. This is the case of VQRH [105], a novel system that provides the user with several visual representations for both query formulation and result visualization. One possibility is to use 3D features to present the results in the simulated reality setting. For example, if the database refers to the books in a library, a virtual library can be represented in which the physical locations of the books are indicated by icons in a 3D presentation of the book stacks of the library. QBI [54] is another VQS which uses this kind of result presentation [135] and provides a very suggestive visualization, thanks also to the power of specialized graphic workstations.

5. Comparison Based on Interaction Strategies

In Section 3 the process of data retrieval has been decomposed into two main activities: understanding the reality of interest and formulating the query. In the following, we analyse these two activities and classify the VQSs in terms of specific strategies for accomplishing them. Note that some VQSs allow more than one strategy for each activity.

5.1. Understanding the Reality of Interest

A classification of VQSs based on the strategies for understanding the reality of interest is shown in Table 3. Such strategies are grouped into three main classes: (1) top-down, (2) browsing and (3) schema simplification.

Understanding the reality of interest may be a complex task when the database schema is made of hundreds or thousands of concepts and/or the extension of the database is made of millions or billions of instances. What is needed is a mechanism to filter the information considered significant by the user. This may be achieved by means of a *top-down* strategy, where general aspects of the reality are first perceived, and then specific details may be viewed. The top-down strategy is implemented in several ways. The first can be seen as a sequence of iterative refinements, i.e. the system provides for each schema a library of top-down refinements. Each refinement can be obtained from the previous one by means of transformations, which when applied to atomic objects, result in more detailed structures [4].

Table 3. VQS classification according to understanding strategies

Understanding strategy	Systems
Top-down	4, 24, 76, 77
Browsing	
Intensional	4, 52, 57
Extensional	10, 20, 27, 29, 34, 42, 53, 66
Mixed	11, 23, 26, 28, 40, 50, 58, 68, 73
Schema simplification	
View extraction (same model)	4, 21, 22, 59
View extraction (different model)	6, 27, 30, 31, 48, 50, 74

Another way is to provide either selective or hierarchical zoom. In the case of selective zoom, the schema is unique, and the concepts are layered in terms of levels of importance; the schema can be examined at several levels, so that only objects above a specified importance level are visible. The user can also graphically edit the schema, so that irrelevant objects can be removed from the screen. In Figure 10 we see an example session in GUIDE [77]. The schema represents the Current Population survey and contains comprehensive data on individuals, families and households. The entity *person* is the 'focus' concept; in Figure 10(a) only concepts with the highest importance (level 1) are represented, appearing in the schema at a given distance (radius 5) from *person*. Figure 10(b) shows the schema with the importance level set at 3 and radius 4. In the case of hierarchical zoom, the schema is also unique, but the objects may be examined at different hierarchical detail levels. An example of hierarchical zoom is presented in ESCHER [76].

A further strategy is based on the user–system dialogue. The system presented in D'atri *et al.* [24], adopts the universal relation approach [132], whose main issue is query disambiguation, i.e. the translation of an incomplete query into a completely specified one. This corresponds to the selection of the database portion (i.e. the attributes) relevant to the user's request. For this reason, the strategy proposed in D'atri *et al.* [24] consists of a sequence of user's decisions about the attribute relevance (or lack of) with respect to the query interpretation: if the attribute is considered relevant, it is included in the target list.

Another well-established technique for learning about the information content of a schema is *browsing*. Browsing is essentially a viewing technique aimed at gaining knowledge about the database. In principle, it can handle both schemas and instances in a homogeneous way without any distinction [113]. The main hypothesis is that the user has only a minor knowledge about the database and the interaction techniques. Within this hypothesis, the user starts the interaction by examining a concept and its neighborhood (adjacent concepts can be considered as a first level of explanation of the examined concept); then, a new element is selected by the user from neighboring

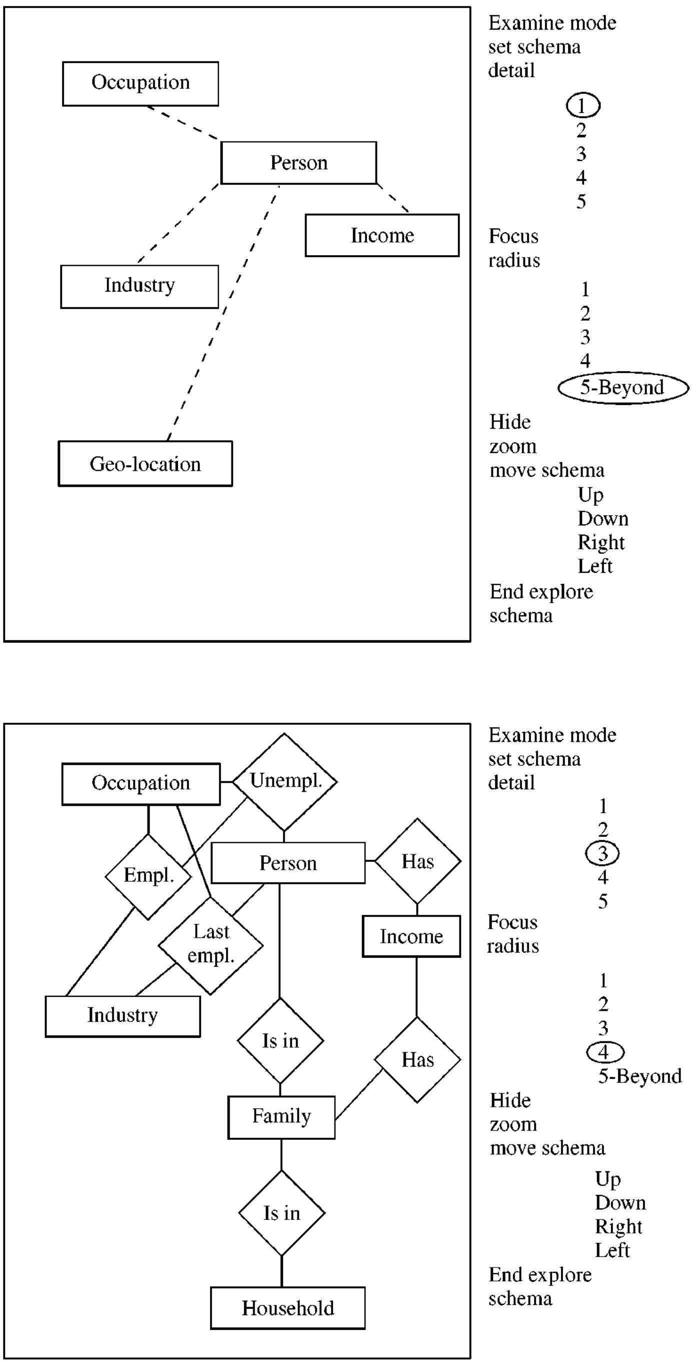


Figure 10. Selective zoom in GUIDE (from Wong and Kuo [77])

concepts to be the current one, and its neighborhood is also shown: this process proceeds iteratively. Thus, an incremental enhancement of the user's knowledge is obtained by exploring logically adjacent concepts. Browsing may be specialized into the following cases:

1. *Intensional.* It is performed on the schema of the database, i.e. on the conceptual schema. The approach, followed for example in QBD* [4], corresponds to metaquerying the schema structure by: (a) asking for all existing paths connecting two concepts and specifying conditions on the length of the path and/or the presence of particular concepts; (b) asking for all the concepts which possess one or more properties; and (c) given a group of concepts, selecting all their neighbor concepts.
2. *Extensional.* It is performed on the extension of the database. Two different approaches may be pointed out. The first, presented for example in G^+ [20], consists of showing the occurrences of the database as vertices of a graph, where edges represent existing links between such occurrences. The second, adopted for example in SUPER [27], starts from the intensional schema of the database, selects an object and browses its instances. A particular kind of browsing is provided by LID (Living In the Database) [34]. The user 'lives' inside a single E–R tuple and sees the database from the perspective of that tuple. Starting a session, the user first selects an entity and chooses, within a list of all entity tuples, the tuple where he wants to reside (called the current tuple). The system then shows the relationships that such tuple participates in as well as the related entities. When a tuple from a linked entity is selected, it becomes the new current one and the display changes according to this new perspective.
3. *Mixed.* In this case the browsing activity may be performed on both the intensional and the extensional part of the database. An example is the KIWI User Interface [26], where possible drawbacks of the browsing paradigms (e.g. inefficiency, 'short-sighted' navigation) are overcome by introducing three models (internal model, interaction model, display model) to represent the information. In particular, the interaction model partitions each object neighborhood into homogeneous subsets and defines suitable structures to ease the presentation of, and the interaction with, the object descriptions.

An alternative approach to top-down refinement and browsing is *schema simplification.* The idea here is to 'bring the schema close to the query'. This is done by building a user view resulting from aggregations and transformations of concepts of the original schema. While in the top-down approach it is possible to locate concepts that exactly match the initial schema (at different levels of abstraction), in the schema simplification approach the user may build a proper view of the original schema which cannot be extracted by the schema itself at any of its levels of abstraction. In the approach followed in QBD* [4], the view data model exactly matches the initial data model. In other approaches (e.g. GORDAS [31]) the view model is different from the initial model so that a better query representation is produced.

5.2. Formulating the Query

Query formulation is the fundamental activity in the process of data retrieval. Table 4 provides a comparison of VQSs based on the different strategies for formulating the query.

Table 4. VQS classification according to query formulation strategies

Query formulation strategy	Systems
By schema navigation	
Arbitrary connected path	2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 16, 23, 26, 27, 34, 36, 39, 41, 43, 44, 46, 47, 50, 51, 57, 58, 60, 62, 66, 68, 72, 76, 78
Connected hierarchical path	27, 30, 31, 35
Unconnected path	4, 18, 21, 22, 24, 28, 33, 40, 49, 64, 65, 74
By subqueries	
Composition of concept	3, 14, 45, 49, 53, 54, 55, 56, 57, 67, 73, 75
Use of stored queries	4, 11, 12, 21, 22, 48, 49, 59, 72, 75, 77, 78
By matching	
By example	25, 32, 38, 61, 69, 71, 79, 80
By pattern	3, 17, 18, 19, 20, 37, 52, 56
By range selection	1, 29, 69

The query strategy *by schema navigation* has the characteristic of concentrating on a concept (or a group of concepts) and moving from it in order to reach other concepts of interest, on which further conditions may be specified. Such a strategy differs according to the type of path followed during the navigation, as described in the following.

1. *Arbitrary connected path.* In this approach, the user navigates along an arbitrary path in the database schema, in order to select the concepts involved in the query. An example is OQL [2], an object-oriented query language, in which the database is represented by means of a semantic diagram. In OQL, a query is expressed by browsing first a semantic diagram of object classes, and then selecting the intensional pattern of interest, i.e. the classes and their associations. At this point, selections are performed by specifying restriction conditions that involve attributes of a single class; inter-class clauses are performed by comparing attributes of two classes.
2. *Connected hierarchical path.* In this case the user focuses on a concept of the database and then the system builds a hierarchical view of the database with the selected concept as the root. For example, in GORDAS the user first selects a root concept that determines the direction of reference for the involved relationships [31]. Relationship attributes are assigned to the entity at the lower level in the hierarchy. Then, the user provides the selection conditions. First, conditions on the attributes of the root entity are specified, and, later on, those involving related entities. It is worth noting that different root entities may be specified for the same query giving rise to different views.
3. *Unconnected path.* In the previous examples the neighbor concepts may be reached starting from the central concept and following paths explicitly represented in the schema. However, the user may be interested in reaching concepts by building new relationships between them. This approach is typical in relational systems, using the so-called outer join operator. For example, in HIQUEL the user starts by selecting an entity name within a list [74]. Then, the system draws onto the screen an empty table for

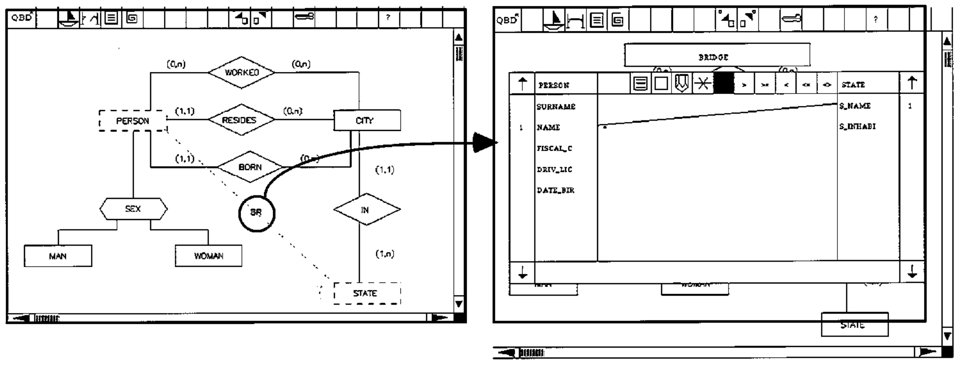


Figure 11. Unconnected path in QBD* (from Angelaccio *et al.* [4])

the entity set, specifying all the attribute names. Step by step, the user builds up the query, first deleting the non-relevant attributes and then writing conditions to select entities and so on. The outer join operator can also be expressed in the Entity Relationship model. For example, in QBD* a query primitive is available that allows joining entities not explicitly linked in the schema [4].

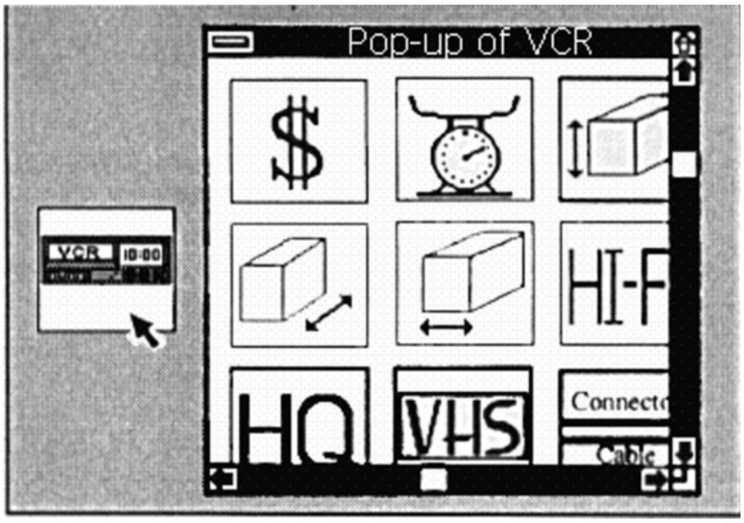
A second strategy for query formulation is *by subqueries*. In this case the query is formulated by composing partial results. Two different approaches are outlined in the following:

1. *Composition of concepts*. This approach is followed in several iconic languages, where the output concepts of the query are produced by composing input concepts. In such a way, queries are specified by overlapping icons. Each operation, such as selecting an icon or overlapping two icons, is interpreted as a particular chain of operations on the database. Figure 12 shows a query formulation in IconicBrowser [73]. The query is: ‘Show the weight of Hi-Fi VCRs (video recorders) with price equal to or cheaper than $600; select the lightest one among them’. The *VCR* icon (a front panel of a typical video recorder) is selected from an icon area and moved to the retrieval area; at this point, the whole icon-set operation for VCRs is shown [see Figure 12(b)]. Next, the $ icon (representing the price) is selected and overlapped onto the *VCR* icon. The condition on the price is then specified by setting a range of values [see Figure 12(a)]. Since a *Hi-Fi* icon appears in the icon-set operation, a further restriction to Hi-Fi VCRs is performed with the same overlapping mechanism. Finally, the icon showing a schematized scale (representing the weight) is superimposed, weights are displayed and the lightest one is selected [Figure 12(b)].
2. *Use of stored queries*. In this case, queries can be composed by using subqueries either stored in a system library or previously computed during the query session. For example, in CGQL [59], subqueries are represented in terms of a parametric skeleton, and, furthermore, it is possible to store the result of a query execution expressed in turn as a so-called named query.

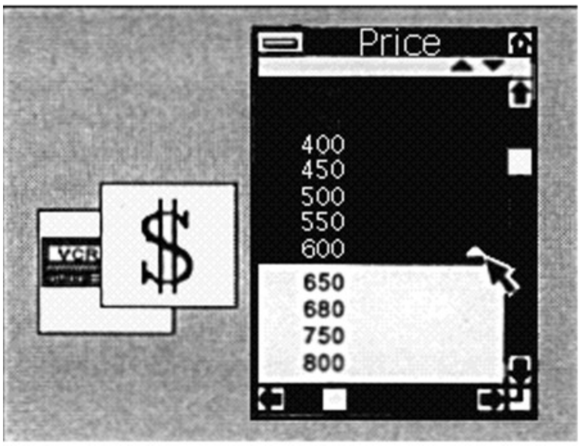
The third strategy for query formulation is *by matching*. It is based on the idea of presenting the structure of a possible answer that is matched against the stored data.

There are two main approaches:

1. *By example.* As an alternative approach to the traditional querying procedure (i.e. the user formulates the request and the system retrieves the answer), the user may provide an example of the answer and the system identifies the goal by generalizing such an example. The most known application of such an approach is QBE [80]. In this querying environment the user specifies a set of tables with several tuples and an example of the requested answer. The system, starting from this example, identifies the query and responds by filling the tables with tuples that match the example. This query paradigm is also followed in QBBE [25], a system based on the idea of integrating

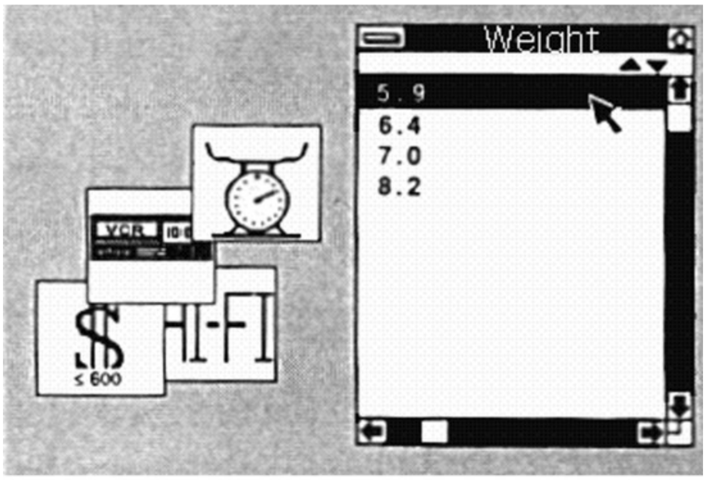


(a)



(b)

Figure 12. Overlapping of icons in IconicBrowser (from Tsuda *et al.* [73])



(c)

Figure 12. Continued

browsing functionalities together with a generalization mechanism for capturing the abstractions needed for constructing queries from examples.

2. *Pattern matching.* In this approach the user provides a pattern and the system looks for all fragments of the database matching such pattern. In G^+ [20] the pattern is expressed by means of a labeled query graph (see top of Figure 13, where the label is a regular expression, used to match the edge labels along simple paths in the graph representing the database). The label $(CA^*NA)^+$ in Figure 13 specifies that we request all the routes of an airline network made of two subpaths, each one in turn made of a set of Canadian Airlines (CA) flights followed by one National Airlines (NA) flight.

The last strategy for query formulation is *by range selection*, allowing a search conditioned by a given range on multi-key data sets to be performed. The query is formulated through direct manipulation of graphical widgets, such as buttons, sliders and scrollable lists, with one widget being used for every key. An interesting implementation of such a technique has been proposed by Ahlberg *et al.* [81], and is called *dynamic query*. The user can either indicate a range of numerical values (with a range slider), or a sequence of names alphabetically ordered (with an alpha slider). Given a query, a new query is easily formulated by moving the position of a slider with a mouse: this is supposed to give a sense of power but also of fun to the user, who is challenged to try other queries and see how the result is modified. Usually, input and output data are of the same type and may even coincide. An application of dynamic queries is shown by Shneidermann *et al.* [70], that refers to a real-estate database. There are sliders for location, number of bedrooms and price of houses in the Washington D.C. area. The user moves these sliders to find appropriate houses. Another interesting application is FilmFinder [1], that allows information about movies to be retrieved by providing names of actors, actresses or movie directors, etc. The user can select some values by using a slider, and this first choice determines the set of values that can be selected by using a so-called tightly coupled slider. This prevents users from specifying null sets. For other uses of dynamic queries, see Catarci and Cruz [97].

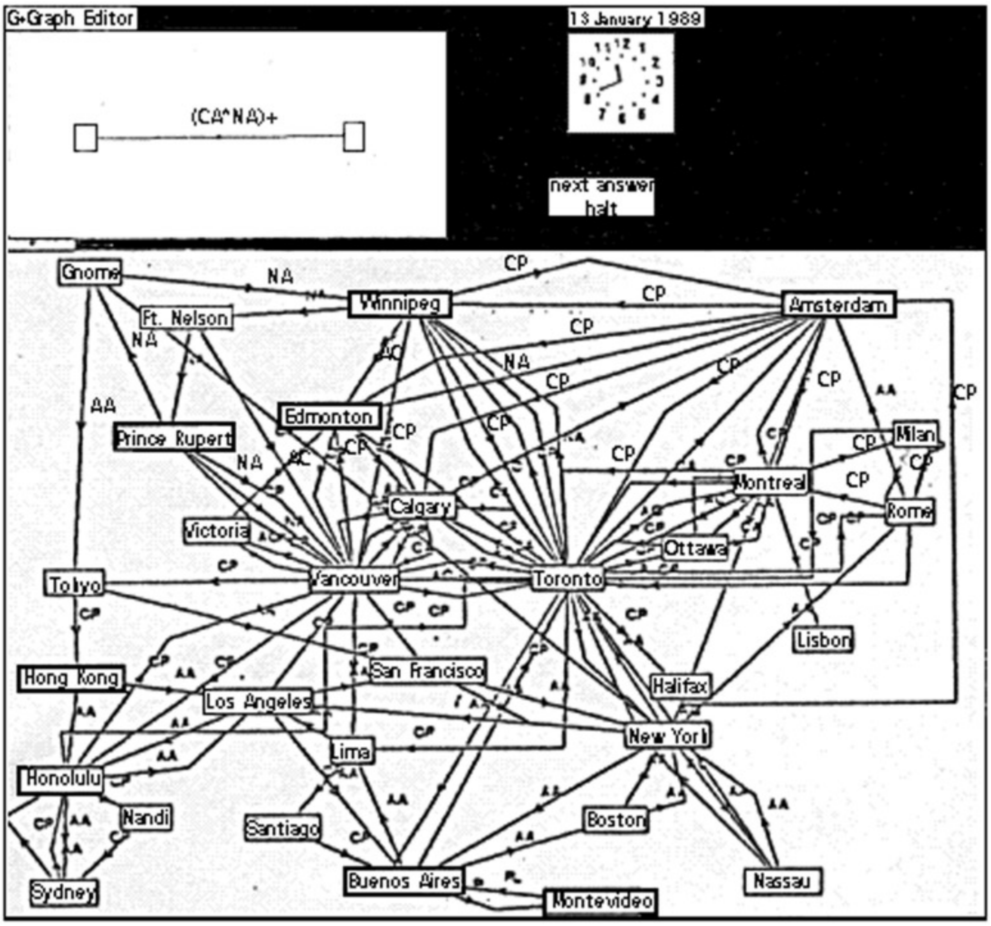


Figure 13. Pattern matching in G^+ (from Cruz *et al.* [20])

6. On VQS Users

Nowadays, while designing interactive systems, the focus of attention has shifted towards the user (the term user-centered has been coined to denote this fact [138]). In order to be user-centered, the development of a VQS, as well as of any interactive system, should be carried out by emphasizing the following three points [84]:

1. the identification of the users and their needs;
2. the usage of this information to develop a system which, through a suitable interface, meets the users' needs;
3. the usability evaluation and validation tests of the system.

None of the VQSs developed so far has completely followed these guidelines. Furthermore, little attention has been given to the range of users that might interact with a system and how the system should adapt to the different types of users.

In order to satisfy the above three points, in this section we first concentrate on some significant features of the potential VQS users, and provide a classification based on their skills and needs. Having analysed different VQSs and classified them, we believe it is important to understand which kind of system may be suitable for each type of user. We focus our analysis on visual representations, the most characterizing aspect of the VQSs, and compare the different representations discussed in Section 4 against the various classes of VQS users. Our conclusions are at least partially supported by the experiments we report in Section 6.2, which have been conducted on some VQS in order to evaluate their usability. Unfortunately, only a few authors report on usability trials made on VQSs.

6.1. User Classes

The type of user that will mostly benefit from the availability of advanced VQSs is the generally called *casual* user. Many authors use this expression, each one stressing one feature as the basic characteristic of this class of users. In Cuff [111], a set of attributes that roughly characterize casual users is provided; unfortunately, no well-characterized group of people emerges. Nevertheless, the features these users have in common help to specify the requirements needed in a new generation of query systems in order to match their needs. Typically, a casual user: (1) interacts with the computer only occasionally, (2) has little, if any, training on computer usage, (3) has a low tolerance for a formal query language and (4) is unfamiliar with the details of the internal organization of an information system.

Since we feel that the above characterization of casual users is still too broad and leads to a class with wide differences among its members, we will try to identify the database users more precisely. Users can be initially distinguished according to the type of computer training they have had during their working activity. *Professional users* have a job which mainly concerns operating computer applications. They possess a wide spectrum of skills on programming languages, database management systems, etc. *Non-professional users* cannot invest time in computer training and usually learn query languages ‘by doing’. It is evident that non-professional users obtain major advantages from easy-to-use query systems that may facilitate an incremental learning process. Therefore, we will concentrate on them in the following, introducing several features that are compared against the basic types of VQS representations discussed in Section 4 (see Figure 14).

Non-professional users can be firstly grouped on the basis of their *frequency of interaction* with the system. The classification corresponding to this criterion is in terms of *occasional* vs. *frequent* users. Iconic VQSs seem the most suitable for occasional users, who could not have learned sophisticated access protocols and therefore, like database representations, closer to the reality they live in. Diagrammatic systems do not appear helpful for very occasional users, since they require learning a specific data model. They become more competitive as the frequency of interaction increases. Since they usually provide a set of predefined queries, form-based systems satisfy the requirements of very frequent users.

The second criterion for classifying non-professional users refers to the *variance of the query* they usually perform. *Repetitive users* tend to express queries having a similar pattern; the similarity may concern either the query structure (e.g. the clerk booking flights in a travel agency) or the types of operators involved in the query (e.g. a statistician often

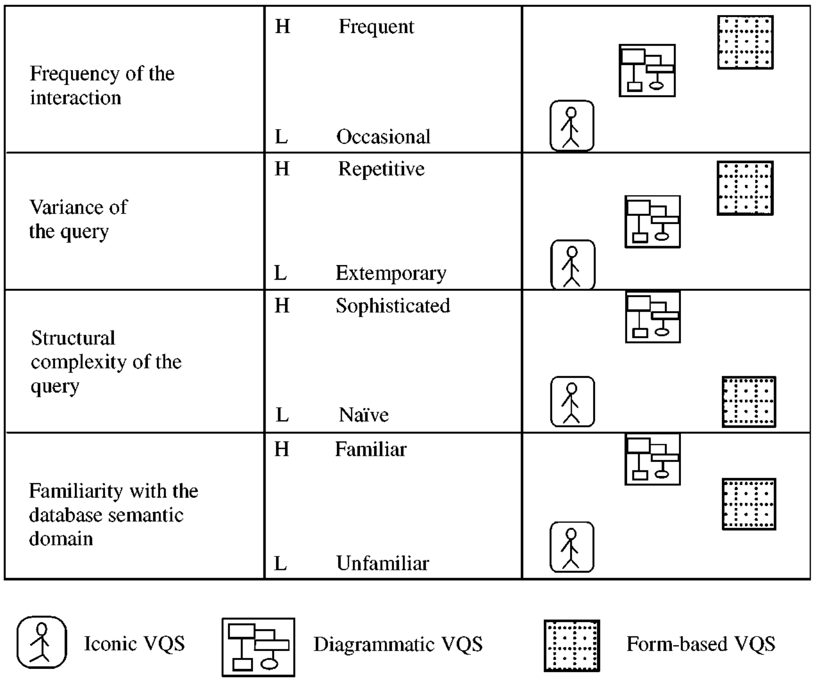


Figure 14. Classification criteria for non-professional users against representation paradigms of VQSs

needs to express aggregations on changeable summary attributes and category attributes). *Extemporary users* have unpredictable needs in terms of the data they want to retrieve. For example, the general manager of a company may need unusual types of correlation between data on production and data on personnel. When the query is well known to the user, the form-based representation looks more suitable, since the access can be performed by simple menus that may denote, by names, a bounded set of queries. For unforeseeable queries, iconic and diagrammatic VQSs are advantageous, since they provide the user with easy navigation in the query space.

The two previous criteria, applied together, give rise to a finer classification that allows one to refer to typical roles in organizations. *Frequent-repetitive users* correspond to administrative clerks and secretaries. Typically, in real systems they interact using menus and forms, and it is up to them to choose a query among a limited predefined set of possibilities.

Occasional-extemporary users tend to delegate the process of query formulation to professional (or more expert) users, since they lack the motivation to spend the time and the intellectual efforts with the perspective of later benefits. VQSs with powerful and friendly training mechanisms can stimulate their curiosity and gradually capture their attention. Among those examined in this survey, a few systems specifically address the needs of these users (see, e.g. Larson and Wallick [51]).

Frequent-extemporary users have usually a managerial role and are motivated to become autonomous in the interaction with the system. The ideal environments for this type of user are flexible yet powerful systems which provide several alternative strategies that the users may choose according to their skills. Finally, *occasional-repetitive users* (i.e. users seldom querying but posing similar requests to the database) do not seem to correspond to meaningful real-life professional figures.

A third criterion, orthogonal to the previous ones, concerns the *structural complexity of the query*. For *naive users* the range of the needed operations is limited, so they may not require the full power of a query language. *Sophisticated users* need to express queries that may require a deep understanding of the underlying computation process and of the query language. Iconic languages are the most appropriate for naive users, since they provide immediate mechanisms to express simple queries. As the structural complexity of the query increases, more powerful operators are needed and, consequently, diagrammatic languages seem to be more suitable.

A final criterion concerns the users' *familiarity with the database semantic domain. Familiar users* are acquainted with the available data, and their only concern is to express their needs using a query language. For example, a traveller who consults a list of flight, bus and train schedules to organize a trip obviously has a clear notion of the concepts of flight, departure time and arrival time but may have strong difficulties in understanding the logical links in the physical representation of all the involved concepts. Typically, in train schedules, departures and arrivals are represented using tables; in case of long routes, the schedule is split into several tables and the logical link among them is expressed by the train code. It is consequently hard to follow the entire route. Conversely, *unfamiliar users* have only a general idea about the contents of the database. Therefore, they need help from the system also to understand the reality of interest. Unfamiliar users greatly gain with an iconic VQS, since they may easily perceive the representation of the reality of interest. This is especially true for simple and widely known semantic domains. Whereas, when the piece of reality involved in the information system is either complex or detailed, the number of icons needed to cover the whole domain increases and, consequently, the discriminating power tends to decrease. In those cases diagrammatic systems, while offering a less immediate representation, do not suffer from such saturation phenomenon, yet form-based VQSs are effective only for elementary domains.

Since the hybrid representation combines all the different characteristics of the VQSs, it provides, in principle, the best visual communication means for the majority of users by supplying different interaction modalities [104]. For instance, recalling the strategies available in SICON [36], the user may interact with both the diagram representing the entire database schema and the individual icons representing specific concepts. In Catarci *et al.* [92], during the process of query expression, the user can actually switch between several different representations, according to his or her preferences and needs.

Since there are a few user taxonomies in the literature, we now briefly compare our classification with the one by Jarke and Vassiliou [124]. These authors distinguish two main criteria, based on *interaction capability*, as a function of familiarity with programming concepts and frequency of system usage, on *task structure*, which is a function of the *application knowledge* (user's knowledge about the contents and also about the structure of the database) and on *range of operations* (how many operations the user requires). These criteria are used to classify all types of users, while we first distinguish among professional and non-professional users and concentrate on the latter, taking into consideration four features. Our feature *structural complexity of the query* actually corresponds to their *range of operation* criterion. We also take into account the *frequency of system usage* (we call it *frequency of interaction*), but we focus on the semantic domain familiarity instead of the application knowledge, since a non-professional user may be aware of what the database is about (i.e. the database contents), but it is very unlikely he or she knows anything about the database structure. Furthermore, we believe the possibility of foreseeing the query is also worth considering, because many non-professional users typically perform repetitive queries. This information may be usefully stored in an advanced query system capable of adapting to the user it is interacting with, as described in by Catarci *et al.* [91], where a system allowing a multiparadigm access to databases by exploiting form-based, diagram-based and icon-based representations is proposed. In Chang *et al.* [106], a further paradigm based on virtual reality has been included in the multiparadigm framework. Another multiparadigm interface is described by Doan *et al.* [116].

6.2. Usability Experiments

Several different definitions of usability exist in the literature. In particular, in Bevan and Macleod [90] a very comprehensive definition of usability is given as 'the extent to which a product can be used with efficiency, effectiveness and satisfaction by specific users to achieve specific goals in specific environments'. More precisely, the effectiveness refers to the extent to which the intended goals of the system can be achieved; the efficiency is the time, the money and the mental effort spent to achieve these goals; the satisfaction depends on how comfortable the users feel using the system. From this point of view, the usability is a major component of the quality of interaction between the user and the overall system.

Usually, at least one measure for each factor determining the quality of interaction must be provided. The effectiveness can be evaluated by relating the goals or subgoals of using the system to the accuracy and completeness that these goals can be achieved with. In the case of VQSs the main goal is to extract information from the database by performing queries, and the accuracy in achieving such a goal is generally measured in terms of the accuracy of query completion (i.e. user's correctness rate when writing the queries). Measures of efficiency relate the level of effectiveness achieved at the expense of various resources, such as mental and physical effort, time, financial cost, etc. In principle, both the user's and the organization's points of view should be considered. However, the user's efficiency is most frequently measured in terms of the time spent to complete a query.

The above two measures (i.e. query accuracy and response time) can be evaluated quite precisely. Frequently, this is done either by recording real users performing predefined tasks with the system and then analysing the recorded data or by directly observing the user. The most common tasks are *query writing* and *query reading*, both of which are performed by investigating the relationships between database queries expressed in natural language and the same queries expressed in the system under study. In query writing, the question is: 'Given a query in natural language how easily can a user express it through the query language statements?' The question for query reading is: 'Given a query expressed through the query language statements, can the user express the query easily in natural language?' Moreover, other kinds of measures can be defined and evaluated, although with less precision.

Measures of satisfaction describe the comfort and acceptability of the overall system used. The learnability of a product may be measured by comparing the usability of a product handled by one user along a time scale. The flexibility of a product can be assessed by measuring usability in different contexts.

Within VQSs, the comparison of QBE against SQL has been mainly reported in the literature [141, 154]. Previous studies [141] showed better user performances when using QBE with respect to SQL, both in query reading and query writing tests. However, in Yen and Scamell [154] QBE and SQL were again compared, taking into account several factors, such as the use of the same database management system, a similar environment, etc. It is interesting to note that the query language type affected user performance only in 'paper and pencil' tests, in which case QBE users have higher scores than SQL users. In on-line tests, the user's accuracy was not affected by the type of the language adopted, but the user's satisfaction was much greater with QBE, and his or her efficiency much better.

In Ahlberg *et al.* [81], a language based on *dynamic queries* (see Section 5.2) was tested against two other query languages, both providing form fill-in as the input method. One of these languages (called FG) has a graphical visualization output, and the other one (FT) has a fully textual output. The alternative interfaces were chosen to find out which aspect of dynamic queries makes the major difference, either the input by sliders, or the output visualization. The tasks to be performed by the user concerned basically the selection of elements which satisfy certain conditions. However, the subjects were also asked to find a trend for a data property and to find an exception for a trend. The hypothesis that the dynamic queries language would perform better than both the FG and the FT interface was confirmed. Similarly, the FG interface produced faster completion times than the FT interface. In particular, for the task of finding a trend, the possibility of getting an overview of the database (in the dynamic and FG interface) made the major difference. In searching for an exception, the dynamic interface performed significantly better than the FG and FT ones. This was due to the advantages offered by both the visualization and the sliders. The visualization allowed subjects to see exceptions easily when they showed up on the screen, and the sliders allowed them to quickly change the values to find the correct answer.

Recently, some experiments have been conducted (see Badre *et al.* [85] and Catarci and Santucci [98]) to compare a diagrammatic query language, namely QBD* [4], against both SQL and QBI [54], an iconic query language. The overall objective of the studies was measuring and understanding the comparative effectiveness and efficiency with which subjects can construct queries in SQL or in the diagrammatic or iconic languages. The experiments were designed to determine if there is a significant correlation between (1) the query class and the query language type and (2) the type of query language and the experience of the user. The subjects were undergraduate students, secretaries and professionals having different levels of expertise. The results of the comparison between QBD* and SQL confirmed the intuitive feeling that a visual language is easier to understand and use than a traditional textual language not only for novice users, but also for expert ones. The experts' errors when using SQL were mainly due to the need of remembering table names and using a precise syntax. Working with QBD*, users can gain from looking at the E-R diagrams. Davis proved the advantage of the availability of E-R diagrams for users with some database background, as opposed to traditional textual representations [114].

On the basis of the figures we obtained when comparing QBD* and QBI we can say that expert users perform better using the QBD* system, while a small difference exists concerning the performance of non-expert users (slightly better using QBI). We did, however, discover noticeable differences between the class of queries containing cycles (better performed in QBI) and the class of queries containing paths (better performed in QBD*) while considering the query classes. Such differences involve both time and accuracy and are totally independent from the skill of the users. In the second case, the difference could be due to the fact that paths are explicitly represented in the E-R schema (while in QBI there are only sequences of icons) and the user perceives the whole path not as a unique complex function, but as a sequence of single steps. When we considered the queries containing cycles, we noticed that whenever a query involves two attributes belonging to the same concept, the QBD* users became much more confused than the QBI ones. This could be due to the fact that in QBD*, the user navigates in the E-R diagram, and for any concept there is only one graphical symbol, even if it is met twice. While in QBI, when a query contains a concept twice, the user sees two distinct icons on the screen, and is not misled when manipulating the right occurrence of the concept (i.e. the right icon).

7. Formal Aspects

In the last 10 years, the research on VQSs has moved from the realm of ideas to the design of real systems. Most proposals have emphasized the aspects related to user interaction and ease of learning and of use, rather than focusing on formal aspects of query languages, such as syntax, semantics and expressive power. The query languages are usually presented through examples of query formulation, making both the study of language expressiveness and the comparison with other languages difficult. In this survey, we are mainly interested in the user-oriented aspects of VQSs, which give rise to the classification presented in Sections 4 and 5. However, some significant VQSs that have addressed formal issues are worth mentioning.

7.1. Syntax and Semantics

The most popular way of specifying the syntax of traditional programming languages and textual query languages is by using BNF notation. Formal syntax definitions are rare in the case of VQSs and visual languages in general, since the bi-dimensionality of such languages exceeds the capabilities of string-based grammars. The semantics is often operational and is usually expressed by exploiting simple rewriting rules, translating the proposed language into some other well-defined target language. Obviously, any target language has in turn its own semantics, which could be procedural, denotational, algebraic or axiomatic, and this semantics is inherited by the visual language.

Some VQSs take the Relational model as formal counterpart, thus having either the relational algebra or calculus as target language. For example, a QBE-like visualization of relational calculus with set operations is described by Ozsoyoglu and Wang [61], where the syntax of the visual operations is given by referring to equivalent textual constructs. Analogously, in QBD*, a mapping from the graphical operations to a textual language is described and the syntax of the textual language is formally defined [83]. Since the semantics of such a textual language is given in terms of relational algebra, QBD* itself has a relational algebra based semantics. For instance, given a mapping from E-R to

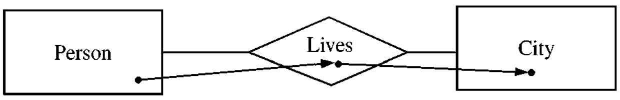


Figure 15. Path selection in QBD*

relational schemata, the selection of the path $\langle$person-lives-city$\rangle$ (see Figure 15) has an equivalent textual format such as: SELECT Person FIND City THROUGH Lives ENDSELECT, which corresponds to the relational algebra expression: $\Pi_{\mathrm{Attr(Person)}}\ \sigma_{\mathrm{C}}(\mathrm{Person} \times \mathrm{Lives} \times \mathrm{City})$.

Many VQSs have some kind of logic as target language, thus exploiting the expressive richness and the sound formalization of logic-based formalisms. In Miura and Moriya [56], two query languages are introduced. The first one, called PIM algebra, is diagram-based, while the second one, PIM calculus, is logic-based. The two languages are shown to have an equivalent expressive power. Formal specifications of the syntax of both languages are also given. The semantics of the graphical operations is specified by logical expressions.

Hy^+ is a hygraph-based query and visualization system [18]. A hygraph is a structure that formally extends the notion of graph by incorporating blobs in addition to edges. A blob relates a containing node with a set of contained nodes, instead of relating a node to another node as edges do. It is possible to assign any semantics to the relationships represented by the blobs, since the meaning will be based on some interpretation of the blob labels. The visual queries supported by the Hy^+ system are expressions of the GraphLog query language [17], suitably extended to hygraphs. A graphical query is a final set of query hygraphs. The semantics of graphical queries is given by a translation to a stratified linear Datalog. Stratified Datalog, extended with universal quantification in rule bodies, is also the reference language for describing the semantics of VQL[a] [75].

Similarly, in G-Log [63], Datalog programs are represented through colored graphs, where the body of the rule is colored red, while the head is colored green. The same directed labeled graphs (without colors) are used to represent database schemata and instances. The nodes of the instance graphs stand for objects, and the edges indicate relationships between values. Whereas, F-logic is used for describing the semantics of DOODLE, a rule-based language that supports user-defined data visualizations and visual queries in an integrated way [19].

A few VQSs refer to programming language constructs for formalizing their behavior. For instance, VQL, described by Mohan and Kashyap [57], is a language for interacting with an object-oriented schema-intensive data model. It consists of a set of graphical primitives that can be combined to create graphical queries. The semantics of the graphical primitives is given as translation to statements of an object-oriented programming language supported by the underlying DBMS. In this case, a graphical syntax simplifies the formulation of some complex queries, that are hardly composed in traditional languages. Figure 16 shows the VQL formulation of the query: ‘Find all those

[a] Note that two different visual query languages exist sharing the same name, i.e. VQL. The first one was proposed by Mohan and Kashyap in 1993 [57], while the more recent one is described by Vadaparty and Subrahmanyam [75].

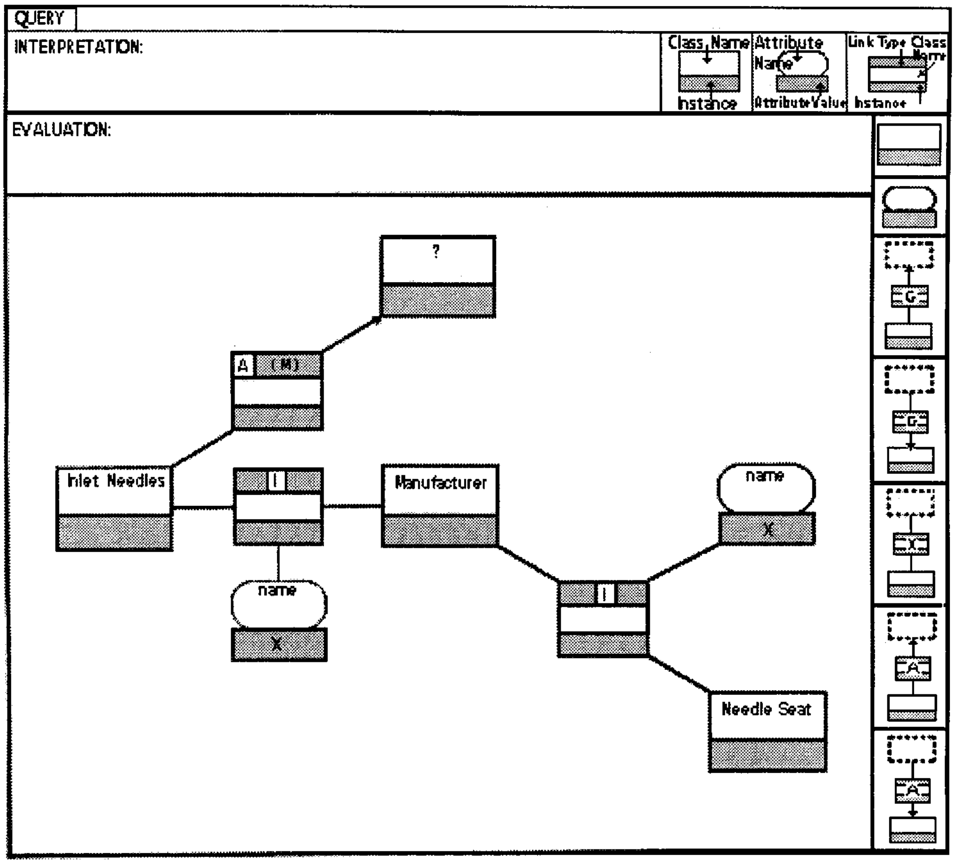


Figure 16. Query formulation in VQL (from Mohan and Kashyap [57])

assemblies that use needles that are manufactured by manufacturers who also make seats'. Such a query is built by selecting and assembling pre-defined graphical primitives, representing classes (e.g. manufacturer), I-link (i.e. interaction link, a notion similar to that of relationship) and attributes. Moreover, the same variable *X* is used in two different attribute primitives, meaning that the value of the variable has to be the same. A syntax and a grammar for combining such primitives are specified. Since a set of textual constructs corresponding to the visual primitives is provided, a formal BNF grammar has been developed for creating valid query expressions in the textual predicate language. This is the way the authors have chosen to indirectly specify a pictorial grammar for combining the visual language primitives themselves. Also, Czejdo *et al.* [21] describe a graphical language for an extended E-R model. Every operator of the language is defined by providing the language name, argument list and semantics, which is given in terms of a set of procedures operating on the database objects and written in a Pascal-like language. Another language for an extended E-R model is HQL/EER [3]. This language has the peculiar characteristic of allowing the user to formulate queries by intermixing both graphical and textual elements. The syntax and the semantics of the hybrid (textual and graphical) sentences is defined formally using programmed graph rewriting systems.

A restricted number of VQSs is provided with a formal semantics without referring to a target language. An operational semantics is directly provided by Catarci and Tarantino [13] for database querying by hypergraph manipulation, where some basic direct-manipulation primitives (IPs) that can be composed and sequenced to build meaningful queries, are defined.

Another recent proposal of a visual query language whose syntax and semantics are clearly stated is by Papantonakis and King [62]. The authors make an effort to provide formal specifications by working directly on the language they have defined, called Gql, and not using an equivalent textual language. Gql is a declarative graphical query language based on the functional data model. The authors' approach to formalization consists in abstracting from the world of graphics and concentrating on a world of sets and functions, called the 'base structure', which describes the various Gql constructs. Since not all possible base structure instances correspond to legal Gql queries, some syntax rules, expressed as truth-value formulae, are imposed on them. The Gql semantics is operational and is given by showing how to translate such instances into list comprehensions which are variant of lambda-calculus expressions and thus have well-defined semantics.

Finally, two papers are worth mentioning, both attempting to provide general formal foundations to diagram-based query languages [99, 100]. In Catarci *et al.* [99] a graphical data model, the Graph Model, in which the visual representation is part of the model itself, and a minimal set of Graphical Primitives, in terms of which general query operations may be visually expressed, are proposed. The Graph Model and the Graphical Primitives have both a visual syntax and a set-theoretical semantics. They can be easily used as basic constituents of more complex existing visual representations and visual query languages, thus giving them a formal semantics independent of the underlying data model, and offering a common comparison framework. Analogously, in Catarci and Tarantino [100], a theoretical framework for visual interaction with databases is presented, having visual syntax and operational semantics. The main feature of such an approach is the adoption of a particular kind of hypergraph, the *structure modeling hypergraph* (*SMH*), as a powerful representation tool, able to capture the features of existing data models. Notable characteristics of SMHs are: uniform and unified representation of intensional and extensional aspects of databases, direct representation of containment relationships and immediate applicability of direct manipulation primitives. SMHs are not a new data model but a new representation language that provides the syntactic rules for describing the structuring mechanisms of data models. SMHs can be queried by formal systems closed under queries. As such, they represent a formal tool against which existing VQLs could be compared by expressing their visual primitives in terms of the SMH query system.

7.2. Expressive Power

The expressive power of a query system, i.e. the ability of the system to extract information from the database, is also a basic aspect in VQSs, as stressed by Aho and Ullman [82]. Meaningful classes of queries have been investigated by Chandra [102], giving rise to the so-called Chandra's hierarchy. Among the different VQSs referred to in the literature, only a few have been formally characterized in terms of the expressive power of their query languages. In this subsection we briefly discuss an enlarged Chandra's hierarchy and show the position of some VQSs inside the different classes of queries (see Batini *et al.* [86] for a deeper discussion on the subject).

A significant class of queries inside Chandra's hierarchy, namely the class of *first-order queries*, was due to Codd [109]. First-order logic seemed fairly expressive, and the term *completeness* was used to indicate that a query language could express all first-order queries. However, it soon became apparent that the amount of expressive power provided by such a language was not adequate to express often useful queries such as transitive closures. For example, Zloof first considered adding a primitive for transitive closure to QBE [158]. Among classes with a higher expressive power, *fixpoint queries* aim at enriching relational algebra with the fixpoint operator, thus allowing, as a subcase, to express transitive closures of binary relations. The class of *Horn clause queries* is based on the logical paradigm, and is not comparable with relational algebra, since queries can be found in either class that cannot be expressed in the other class. For instance, in a database containing information on people, parents and spouses, one can ask for the ancestors of a person by using a Horn clauses-based language (this query cannot be expressed in relational algebra), whereas relational algebra allows to obtain the list of unmarried people and Horn clauses do not. An extension of the Horn clause queries yields *stratified* queries that can express all first-order queries and are a proper subset of fixpoint queries (fixpoint queries are obtained by augmenting the standard relational algebra with fixpoints). Stratified queries include not only first-order and transitive closure queries, but also other interesting queries, such as those corresponding to the 'path accessibility problem' on boolean circuit families. Furthermore, finding fixpoint queries that are not computable through a stratified Datalog program is a hard task. It seems that the only fixpoint queries (on finite structures, such as databases) which are not expressible by stratified logic programs are related to the need of taking fixpoints over universal quantifiers, as it happens in the problems of determining winning strategies between two or more players in a certain game [128], and this could be considered a marginal need from the point of view of a potential user of a query system. Then, stratified Datalog seems to be largely considered as a new benchmark in the theory of the query language expressive power (as it was for the relational algebra in the recent past). Nevertheless, languages that friendly express queries less than first order are also very important, since the majority of queries used in database systems is simple.

As a consequence, the position of the existing visual query languages inside Chandra's hierarchy is rather unbalanced. On the one hand, there is a growing attempt to design powerful VQSs, which could be as much powerful as stratified Datalog is, but much easier to use, resulting in a populated area around the class of fixpoint queries. On the other hand, most visual query languages developed so far do not have a high expressive power, since they are directed at casual users, mainly interested in friendly expressing simple queries. They are as (see Alashqur *et al.* [2], Benzi *et al.* [6], Clark and Wu [16], Dennebouy *et al.* [27], Embley [32], Hild and Poulovassilis [40], Kangassalo [44], Larson and Wallick [51], Nanni [59], Orman [60], Papantonakis and King [62] and Rogers and Cattell [68]) or less (e.g. Carey *et al.* [11], Elmasri and Larson [30], Groette and Nilsson [36] and Wong and Kuo [77]) expressive as relational algebra, and very few are placed in the upper levels of the hierarchy.

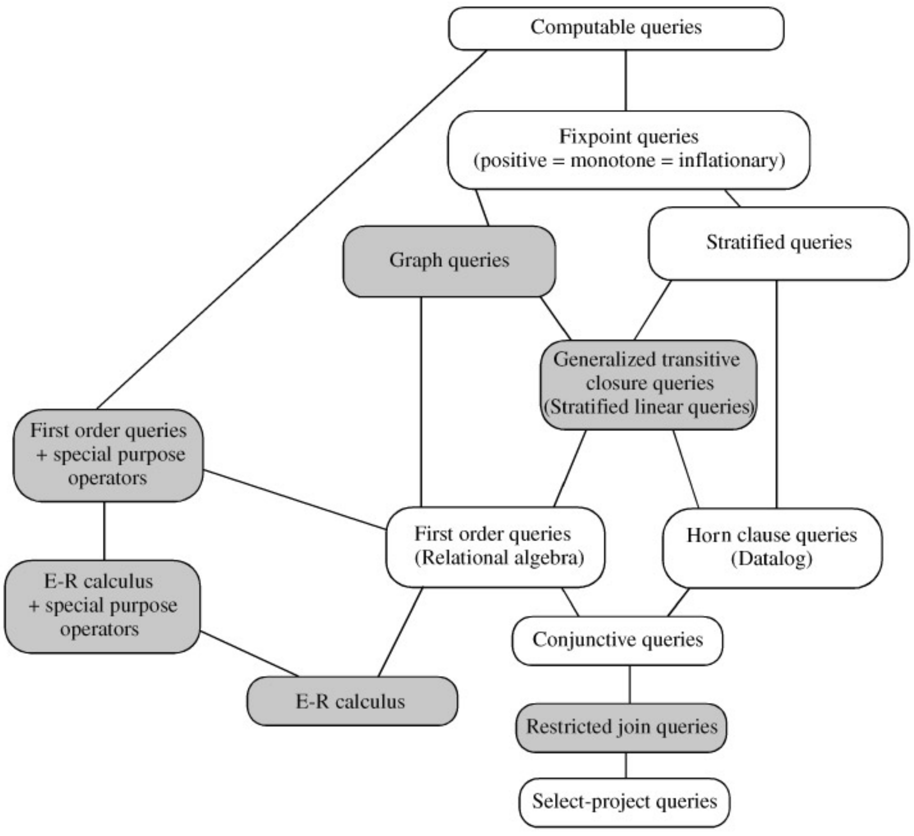

Computable queries
Fixpoint queries (positive = monotone = inflationary)
Graph queries
Stratified queries
Generalized transitive closure queries (Stratified linear queries)
First order queries + special purpose operators
First order queries (Relational algebra)
Horn clause queries (Datalog)
E-R calculus + special purpose operators
Conjunctive queries
E-R calculus
Restricted join queries
Select-project queries

Figure 17. Classes of queries in the enriched Chandra's hierarchy

In Figure 17 we show a more detailed hierarchy which has been enriched by the addition of six new classes (shown with gray hatched boxes):

1. *Generalized transitive closure queries.* They correspond to the class of queries obtained by extending the relational algebra with the generalized transitive closure operator [151], a transitive closure operation where cycle conditions are extended to be Boolean expressions with equality and inequality operators. They have been proven to be equivalent to *stratified linear* queries and to non-deterministic logarithmic space queries (assuming an ordering on the domain) [17]. VQSs falling in this class are the work of Angelaccio *et al.* [4], Cruz [19], Mohan and Kashyap [57] and Zloof [80].
2. *Graph queries.* They correspond to the queries expressible by using the diagrammatic query languages Graphlog [17] and G^+ [20], and involve the computation of simple paths in directed graphs.
3. *First-order queries enriched with special purpose operators.* They are obtained by enriching relational algebra with operators that compute new database values by applying aggregation and counting functions. An example of such queries is: 'Find the global amount of salaries of the employees'. VQSs in this class are described by Cinque *et al.* [15], Sockut *et al.* [72] and Ursprung *et al.* [74].

4. *E-R calculus queries.* They are obtained by using the E-R calculus [5]. The E-R calculus is similar to the relational calculus defined in Codd [109]; the only difference is that atoms containing references to two or more different object sets (either entity sets or relationship sets) are allowed only if the object sets are explicitly related in the schema of interest. See Atzeni and Chen [5], and Elmasri and Larson [30] for examples of this kind of VQSs.

5. *E-R calculus queries enriched with special purpose operators.* Analogous to case 3, they are obtained by enriching E-R calculus operators with aggregation and counting functions. A VQS in this class is the work by Wong and Kuo [77].

6. *Restricted join queries.* They are obtained by using the relational operators *select* and *project* plus a *join* operator, applied to object-classes that must be explicitly related in the database. A VQS allowing such kind of queries is presented by Groette and Nilsson [36].

Attempts have been made to build more powerful VQSs, while retaining at the same time their friendliness. However, the expressive power of such VQSs has not been clearly stated. A first proposal in this direction is G-WHIZ [38], a tabular language for the functional data model [148] in which recursive queries are allowed. Such queries are formulated by means of hierarchical views, i.e. views expressed recursively on both entities and previously defined views. More recently, the visual language Pasta-3 [49] has been proposed. Its expressive power aims at covering the class of computable queries, but, unfortunately, no formal proof for this conjecture is provided. This language allows a simple recursion to be expressed on a single entity by duplicating the entity; the two copies are then linked by the same relationship in two different roles. Unfortunately, the claim of a completely visual system seems to be reduced to the simulation of texts with boxes containing words. A different approach is followed by R^2 [41], where a language for the specification of graphical interfaces is introduced. Using such a language, an interface for nested relational databases is described; the interface allows to express queries in a piecemeal manner by constructing new relations using basic operations from the underlying formalism. A different case is represented by PIM [56], where the semantics of the graphical operation is given by logical expressions. By examining the query language characteristics, it seems that PIM can express first-order queries, while there is no evidence of the deductive capabilities claimed by the authors. Finally, VQL is a language able to express fixpoint queries [75]. VQL is probably not friendly enough for novice users since it retains a rule-oriented query formulation style, but it is mainly intended to be used as an intermediate layer on top of which application-driven user interfaces should be built.

While traditional query languages have been normally compared with each other in terms of their expressive power, a similar analysis has rarely been performed for visual query languages. The languages adopted in QBD* [4] and G^+ [20] have been compared by Angelaccio *et al.* [83], and it has been proven that they both include the class of languages that allow generalized transitive closure queries.

As a final remark, we must take into account that in several cases the user's queries are not precise and specific [137]. Even sophisticated users could express a request with a vague or imprecise condition, such as: retrieve all suppliers of 'small' parts in 'recently introduced' products. Most conventional database systems cannot handle vague queries directly. Whereas, VQSs are usually equipped with mechanisms, e.g. browsers, for performing exploratory searches, so to help the users to specify their queries incrementally. While browsing, users gain insight into the contents and organization of the database. Furthermore, powerful visualizations provide the feature of filtering data from overview displays by means of simple mechanisms such as zooming. The user starts visualizing the whole data set and then modifies this view to show the data portion of interest [1, 70].

In Chang *et al.* [105, 106], the concept of *progressive query* is proposed: the user, who is not able to compose directly a query whose result fully satisfies his or her needs, can formulate the query progressively, i.e. step by step, by first asking general questions, obtaining preliminary results and then revisiting such outcomes to direct the query further in order to extract the result he or she is interested in. Note that the appropriate visualization of the preliminary result provides a significant feedback to the user and a useful suggestion about the right way to proceed towards the ultimate query.

8. Open Problems

The goal of VQS design is to build interfaces that allow users to easily communicate their requests and obtain satisfactory answers. Since different users have different preferences and should not be coerced by the system, any VQS should also provide different ways for expressing the queries and eventually adapt to each single user. This general objective of creating friendly and powerful query systems has further implications. For instance, the system interface should allow any user to interact with the system without any previous training. The query language should be powerful enough to permit, in principle, all computable queries to be expressed, and any type of data should be retrieved. Implicit information should be deduced so that the system could always provide the user with the right answer or declare its ignorance about a certain fact. Moreover, in order to be sure that the query has been successfully answered, the system should be provided with some metrics for the user's satisfaction. Unfortunately, various open problems exist in presently available systems with respect to the above issues, currently investigated by several researchers. Let us conclude this work by scanning the issues which appear to be the most significant for the design of the next generation of VQSs (note that some of them are peculiar to VQSs, while other ones are common to the generality of query systems).

1. Further improvement of the available visual representations by using the third dimension, animation and virtual reality features.
2. Extension of available data types, like natural language texts, geographic maps, images and sounds.
3. Generation of a user model that formally describes the user's interests and skills and that should be dynamically maintained by the system according to the analysis of the story of interactions.
4. Definition of appropriate metrics and methodologies to evaluate the system usability.
5. More experiments of the system, both prototypes and final version, with real users validating it and evaluating its usability.
6. Provision of a formal semantics.
7. Enrichment of the expressive power towards more powerful classes in Chandra's hierarchy.

8. Definition of a different query hierarchy based on the concept of the ‘usefulness’ of the various queries for users; it could then be compared against Chandra’s hierarchy to find the most significant intersections.
9. Use of mechanisms for inferring implicit knowledge from stored facts.
10. Handling of indefinite knowledge, metaknowledge, multiple agents, approximate questioning (where queries are allowed that do not specify exactly the desired result), etc.

The above issues are becoming more and more crucial given the enormously growing diffusion of network communication, e.g. the World Wide Web and digital commerce, both of which connect any type of users. Such users need to locate several types of information, ranging from unstructured documents and pictures to structured data residing in database management systems. The access mechanisms provided for the various types of information are very different, but need to be integrated into homogeneous systems in order to allow people to fully get hold of the Internet information treasure. VQSs, equipped with visual representations of data schemata and instances, as well as direct manipulation query mechanisms and browsing facilities can be profitably used also in accessing data sources through the Internet. However, this will require further developments in all the above issues, together with retrieval tools able to overcome the existing dichotomy between browsing and querying with respect to both structured and unstructured data.

In addition, building effective metaphors will ease the user’s learning effort, help them to work out a mental model of both data and query and enable them to understand fully the software working at the interface and application levels, allowing them to easily perform correct queries. Finally, other aspects which are significant for the design of successful VQSs are related to technological progress, with particular reference to the evolution of special purpose fast processors, extended memory planes devoted to color, reconfigurable architectures, etc. New interactive devices like sophisticated mice, gestural and tactile sensors and other pointing tools will allow to integrate into a multimedia system the possibilities for an improved human–computer communication which is oriented towards the expression of simple, natural and correct queries which will place the human beings at the center of the system design process.

Acknowledgements

We would like to thank the anonymous referees for their extremely helpful comments. Thanks are also due to Paola Loizzo and Santo Bellino for their help in preparing the classification tables.

This work has been partially supported by MURST 40%.

References

1. C. Ahlberg & B. Shneiderman (1994) Visual information seeking: tight coupling of dynamic query filters with starfield displays. In: *Procedings of the ACM Conference on Human Factors in Computing Systems CHI’94*, Boston, U.S.A., pp. 313–317.

2. A. M. Alashqur, S. Y. W. Su & H. Lam (1989) OQL—a query language for manipulating object-oriented databases. In: *Proceedings of the XV VLDB Conference*, Amsterdam, pp. 434–442.
3. M. Andries & G. Engels (1996) A hybrid query language for an extended entity-relationship model. *Journal of Visual Languages and Computing* 7, 321–352.
4. M. Angelaccio, T. Catarci & G. Santucci (1990) QBD*: A fully visual query system. *Journal of Visual Languages and Computing* 1, 255–273.
5. P. Atzeni & P. P. Chen (1981) Completeness of query languages for the entity relationship model. In: *Proceedings of the 2nd International Conference on the Entity Relationship Approach to Information Modelling and Analysis* (P.P. Chen, ed.), Washington, DC, North Holland, Amsterdam, pp. 111–124.
6. F. Benzi, D. Maio & S. Rizzi (1996) Visionary: a visual query language based on the user viewpoint approach. In: *Electronic Series Workshop in Computing.* Springer, London.
7. A. Berztiss (1990) The visual languages Quela and Vizla. Technical Report Number 168, Department of Computer and Systems Sciences, The Royal Institute of Technology and Stockholm University, Sweden.
8. J. Boyle, S. Leishman & P. M. D. Gray (1996) From WIMP to 3D: the development of AMAZE. *Journal of Visual Languages and Computing* 7, 291–319.
9. D. Bryce & R. Hull (1986) SNAP: a graphics-based schema manager. In: *Proceedings of the IEEE Conference on Data Engineering*, Los Angeles, U.S.A., pp. 151–164.
10. L. M. Burns, A. Malhotra, G. Sockut & K. Y. Whang (1991) AERIAL: ad hoc entity-relationship investigation and learning. In: *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics.*
11. M. Carey, L. Haas, V. Maganty & J. Williams (1996) PESTO: an integrated query/browser for object databases. In: *Proceedings of the XXII VLDB Conference*, Bombay, pp. 203–214.
12. T. Catarci & G. Santucci (1989) GRASP: a graphical system for statistical databases. In: *Proceedings of the 1989 SSDBM Conference*, Charlotte, North Carolina, U.S.A., pp. 401–406.
13. T. Catarci & L. Tarantino (1994) Database querying by hypergraph manipulation. In: *Interfaces to Database Systems* (P. Sawyer, ed.), *Series Workshop in Computing.* Springer, London, pp. 84–103.
14. S. K. Chang, M. J. Tauber, B. Yu & J. S. Yu (1989) A visual language compiler. *IEEE Transaction on Software Engineering* 15, 506–525.
15. L. Cinque, S. Levialdi & F. Ferloni (1991) An expert visual query system. *Journal of Visual Languages and Computing* 2, 101–113.
16. G. J. Clark & C. T. Wu (1994) DFQL: dataflow query language for relational databases. Information and management 27, 1–15.
17. M. P. Consens & A. O. Mendelzon (1990) GraphLog: a visual formalism for real life recursion. In: *Proceedings of the 9th ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, pp. 404–416, Noshville, Tennessee, April 2–4, 1990, ACM Press, NY, U.S.A.
18. M. P. Consens & A. O. Mendelzon (1993) Hy$^+$: a hygraph-based query and visualization system. In: *Proceedings of the ACM SIGMOD Conference on Management of Data*, pp. 511–516.
19. I. F. Cruz (1992) DOODLE: a visual language for object-oriented databases. In: *Proceedings of the ACM SIGMOD Conference on Management of Data*, San Diego, California, June 2–5, pp. 71–80. M. Stonebraker (Ed.), SIGMOD Record 21(2), June 1992.
20. I. F. Cruz, A. O. Mendelzon & P. T. Wood (1988) G$^+$: recursive queries without recursion. In: *Proceedings of the 2nd International Conference on Expert Database Systems*, pp. 355–368.
21. B. Czejdo, R. Elmasri, D. Embley & M. Rusinkiewicz (1990) A graphical data manipulation language for an extended entity-relationship model. *IEEE Computer* 23, pp. 26–36.
22. B. Czejdo, D. Embley, V. Reddy and M. Rusinkiewicz (1989) A visual query language for an E-R data model. In: *Proceedings of the International Workshop on Visual Languages*, Roma, Italy, pp. 165–170.
23. S. Dar, N. H. Gehani, H. V. Jagadish & J. Srinivasan (1995) Queries in an object-oriented graphical interface. *Journal of Visual Languages and Computing* 6, 27–52.
24. A. D'atri, P. Di Felice & M. Moscarini (1989) Dynamic query interpretation in relational databases. *Information Systems* 14, 195–204.
25. A. D'atri, R. Menghini & L. Tarantino (1991) Query by browsing and examples. Unpublished manuscript.

26. A. D'atri & L. Tarantino (1989) A friendly graphical environment for interacting with data and knowledge bases. In: *Proceedings of the 3rd Human-Computer Interaction Conference*, Boston, U.S.A., pp. 211–218.
27. Y. Dennebouy, M. Andersson, A. Auddino, Y. Dupont, E. Fontana, M. Gentile and S. Spaccapietra (1995) SUPER: visual interfaces for object + relationship data models. *Journal of Visual Languages and Computing* 6, 74–99.
28. O. Deux *et al.* (1990) The story of O_2. *IEEE Transactions on Knowledge and Data Engineering* 2, 91–108.
29. G. P. Ellis, J. E. Finley & A. S. Pollitt (1994) HIBROWSE for hotels: bridging the gap between user and system views of a database. In: *Interfaces to Database Systems* (P. Sawyer, ed.), *Series Workshop in Computing*. Springer, London, pp. 49–62.
30. R. Elmasri & J. A. Larson (1985) A graphical query facility for ER databases. In: *Proceedings of the 4th International Conference on Entity-Relationship Approach*, Chicago, Illinois, U.S.A., pp. 236–245.
31. R. Elmasri & G. Wiederhold (1981) GORDAS: a formal high-level query language for the entity-relationship model. In: *Proceedings of the 2nd International Conference on Entity-Relationship Approach*, Washington, DC, U.S.A., pp. 49–72.
32. D. W. Embley (1989) NFQL: the natural forms query language. *ACM Transactions on Database Systems* 14, 168–211.
33. R. G. Epstein (1991) The TableTalk query language. *Journal of Visual Languages and Computing* 2, 115–141.
34. D. Fogg (1984) Lessons from a 'living in a database' graphical query interface. In: *Proceedings of the ACM-SIGMOD Conference on the Management of Data*, pp. 100–106. Boston, Massach., June 18–21, 1984 (B. Yormark, ed.).
35. K. J. Goldman, S. A. Goldman, P. C. Kanellakis & S. B. Zdonik (1985) ISIS: interface for a semantic information system. In: *Proceedings of the ACM-SIGMOD Conference on the Management of Data*, pp. 328–342.
36. I. P. Groette & E. G. Nilsson (1988) SICON: an icon presentation module for an E-R database. In: *Proceedings of the 7th International Conference on Entity-Relationship Approach*, Roma, Italy, pp. 271–289.
37. M. Gyssens, J. Paredaens & D. Van Gucht (1990) A graph oriented object model for database end-user interfaces. In: *Proceedings of the ACM-SIGMOD Conference on the Management of Data*, Atlantic City, U.S.A., pp. 24–33.
38. S. Heiler & A. Rosenthal (1985) G-WHIZ, a visual interface for the functional model with recursion. In: *Proceedings of the 11th International Conference on Very Large Databases*, Stockolm, pp. 209–218.
39. P. Hietala & J. Nummenmaa (1993) MEDUSA—a multimodal database user interface and framework supporting user learning and user interface evaluation. In: *Interfaces to Database Systems* (R. Cooper, ed.), *Series Workshop in Computing*. Springer, London, pp. 392–405.
40. S. Hild & A. Poulovassilis (1996) Implementing hyperlog, a graph-based database language. *Journal of Visual Languages and Computing* 7, 267–289.
41. G. Houben & J. Paredaens (1989) A graphical interface formalism: specifying nested relational databases. In: *Visual Database Systems* (T. L. Kunji, ed.). North-Holland, Amsterdam.
42. R. Inder & J. Stader (1994) Bags and viewers: a metaphor for structuring a database browser. In: *Interfaces to Database Systems* (P. Sawyer, ed.), *Series Workshop in Computing*. Springer, London, pp. 215–235.
43. H. Kangassalo (1988) COMIC: a system for conceptual modelling and information construction. Technical Report, University of Tampere, Finland.
44. H. Kangassalo (1988) CONCEPT D: a graphical language for conceptual modelling and data base use. In: *Proceedings of the IEEE Workshop on Visual Languages*, Pittsburgh, U.S.A., pp. 2–11.
45. H. J. Kim, H. F. Korth & A. Silberschatz (1988) PICASSO: a graphical query language. *Software Practice and Experience* 18, 169–203.
46. R. King & S. Melville (1984) SKI: a semantic knowledgeable interface. In: *Proceedings of the 10th International Conference on Very Large Databases* Singapore, pp. 30–37.

47. H. -J. Klein & D. Kramer (1995) NQS—a graphical query system for data models with binary relationship types. In: *Proceedings of the IFIP 2·6 Third Working Conference on Visual Database Systems (VDB-3)*, Lausanne, Switzerland, pp. 349–363.
48. M. Kuntz (1992) The gist of GIUKU graphical interactive intelligent utilities for knowledgeable users of data base systems. *Sigmod Record* 21, 58–64.
49. M. Kuntz & R. Melchert (1989) Pasta-3's graphical query language: direct manipulation, cooperative queries, full expressive power. In: *Proceedings of the Fifteenth International Conference on Very Large Data Bases* North-Holland, Amsterdam.
50. J. A. Larson (1986) A visual approach to browsing in a database environment. *IEEE Computer* 19, 62–71.
51. J. A. Larson & J. B. Wallick (1984) An interface for novice and infrequent database management systems users. In: *Proceedings of the National Computer Conference.*
52. L. Mark (1989) A graphical query language for the binary relationship model. *Information Systems* 14, 231–246.
53. P. Marti, M. Profili, P. Raffaelli & G. Toffoli (1992) Graphics, hyperqueries, and natural language: an integrated approach to user-computer interfaces. In: *Proceedings of the International Workshop on Advanced Visual Interfaces AVI'92.* World Scientific, Singapore, pp. 68–84.
54. A. Massari & P. K. Chrysanthis (1995) Visual query of completely encapsulated objects. In: *Proceedings of the 5th International Workshop on Research Issues on Data Engineering*, Taipei, Taiwan, pp. 18–25.
55. N. Mcdonald & M. Stonebraker (1975) CUPID—the friendly query language. In: *Proceedings of the ACM Pacific 75 Conference*, pp. 127–131.
56. T. Miura & K. Moriya (1992) On the completeness of visual operations for a Semantic data model. *Data and Knowledge Engineering* 9, 19–44.
57. L. Mohan & R. L. Kashyap (1993) A visual query language for graphical interaction with schema-intensive databases. *IEEE Transactions on Knowledge and Data Engineering* 5, 843–858.
58. A. Motro, A. D'atri & L. Tarantino (1988) The design of KIVIEW: an object oriented browser. In: *Proceedings of the 2nd Conference on Expert Database Systems* (L. Kerschberg, ed.). Benjamin/Cummings Publishing Company, Vienna, Virginia, U.S.A., pp. 107–133.
59. U. Nanni (1987) A graphic interface for relational databases. Technical Report, University of Roma, Italy.
60. L. Orman (1992) A visual data model. *Journal of Data and Knowledge Engineering* 7, 227–238.
61. G. Ozsoyoglu & H. Wang (1989) A relational calculus with set operators, its satefy and equivalent graphical languages. *IEEE Transactionss on Software Engineering* 15, 1032–1058.
62. A. Papantonakis & P. J. H. King (1995) Syntax and semantics of Gql, a graphical query language. *Journal of Visual Languages and Computing* 6, 3–25.
63. J. Paredaens, P. Peelman, & L. Tanca (1995) G-Log: a graph-based query language. *IEEE Transactions on Knowledge and Data Engineering* 7, 436–453.
64. E. Pichat & D. Saker (1995) An automatic and cooperative visual database interface. In: *Proceedings of the IFIP 2·6 3rd Working Conference on Visual Database Systems* (VDB-3), Lausanne, Switzerland, pp. 333–348.
65. H. B. Ramos (1993) Design and implementation of a graphical SQL with generic capabilities. In: *Interfaces to Database Systems* (R. Cooper, ed.), *Series Workshop in Computing.* Springer, London, pp. 74–91.
66. M. H. Rapley & J. B. Kennedy (1994) Three dimensional interface for an object oriented database. In: *Interfaces to Database Systems* (P. Sawyer, ed.), *Series Workshop in Computing.* Springer, London, pp. 143–167.
67. W. F. Riekert (1987) The ZOO metasystem: a direct-manipulation interface to object-oriented knowledge bases. Technical Report, Research Group INFORM, University of Stuttgart, Germany.
68. T. R. Rogers & R. G. Cattell (1987) Entity-relationship database user interfaces. In: *Proceedings of the 6th International Conference on Entity-Relationship Approach.* New York, U.S.A., pp. 323–335.
69. Y. Shirota, Y. Shirai & T. L. Kunii (1989) Sophisticated form-oriented database interface for non-programmers. In: *Visual Database Systems* (T. L. Kunji, ed.). North-Holland, Amsterdam, pp. 127–155.

70. B. Shneidermann, C. Williamson & C. Ahlberg (1992) Another queries: database searching by dyrect manipulation. In: *Proceedings of the ACM Conference on Human Factors in Computing Systems CHI '92*. Monterey, CA, pp. 669–670.
71. N. C. Shu (1985) FORMAL: a forms-oriented, visual-directed application development system. *IEEE Computer* 18, 38–49.
72. G. H. Sockut, L. M. Burns, A. Malhotra & K. Y. Whang (1993) GRAQULA: a graphical query language for entity-relationship or relational database. *Data and Knowledge Engineering* 11, 171–202.
73. K. Tsuda, A. Yoshitaka, M. Hirakawa, M. Tanaka & T. Ichikawa (1990) Iconic browser: an iconic retrieval system for object-oriented databases. *Journal of Visual Languages and Computing* 1, 59–76.
74. P. Ursprung & C. A. Zehnder (1983) HIQUEL: an interactive query language to define and use hierarchies. In: *Proceedings of the 3rd International Conference on Entity-Relationship Approach.*
75. K. Vadaparty & R. Subrahmanyam, Towards a formal basis for graphical user-interfaces for relational/object databases. Technical Report 9505, Case Western Reserve University, Cleveland, OH 44106, U.S.A., 1995.
76. L. M. Wegner (1989) ESCHER: interactive, visual handling of complex objects in the extended NF^2-database model. In: *Visual Database Systems.* (T. L. Kunji, Ed.). North-Holland, Amsterdam, pp. 277–297.
77. H. K. T. Wong & I. Kuo (1982) GUIDE: graphical user interface for database exploration. In: *Proceedings of the VIII VLDB Conference*, Mexico City, pp. 22–31.
78. Z. Q. Zhang & A. O. Mendelzon (1983) A graphical query language for entity-relationship databases. In: *Proceedings of 3rd International Conference on Entity-Relationship Approach*, pp. 441–448. Anaheim, California, U.S.A. (C. G. Davis, S. Jajodia, P. A.-B. Ng, R. T. Yeh, eds). North Holland, Amsterdam.
79. J. Zhao, B. Kostka & A. Muller (1993) An integrated approach to task-oriented database retrieval interfaces. In: *Interfaces to Database Systems* (R. Cooper, ed.), *Series Workshop in Computing.* Springer, London, pp. 56–73.
80. M. M. Zloof (1977) Query-by-example: a database language. *IBM System Journal* 16, 324–343.
81. C. Ahlberg, C. Williamson & B. Shneidermann (1992) Dynamic queries for information exploration: an implementation and evaluation In: *Proceedings of the ACM Conference on Human Factors in Computing Systems CHI'92*, Monterey, CA, pp. 619–626.
82. A. V. Aho & J. D. Ullman (1979) Universality of data retrieval languages. In: *Proceedings of the 6th ACM Symposium on Principles of Programming Languages*, pp. 110–120. San Antonio, Texas, January 1979.
83. M. Angelaccio, T. Catarci & G. Santucci (1990) QBD*: a graphical query language with recursion. *IEEE Transactions on Software Engineering* 16, 1150–1163.
84. A. N. Badre (1993) Methodological issues for interface design: a user-centered approach. Technical Report DI/DS-93/01, Università degli Studi di Roma 'La Sapienza'.
85. A. N. Badre, T. Catarci, A. Massari & G. Santucci (1996) Comparative ease of use of a diagrammatic vs. an iconic query language. In: *Electronic Series Workshop in Computing.* Springer, Berlin.
86. C. Batini, T. Catarci, M. F. Costabile & S. Levialdi (1991, revised in 1993) Visual query systems. Technical Report 04.91, Dipartimento di Informatica e Sistemistica, Universita' di Roma 'La Sapienza'.
87. C. Batini, T. Catarci, M. F. Costabile & S. Levialdi (1991) Visual strategies for querying databases. In: *Proceedings of the IEEE Workshop on Visual Language*, pp. 183–189. Kobe, Japan, October 8–11, IEEE Press, Los Alamotos, California.
88. M. Beretta, P. Mussio & M. Protti (1986) Icons: interpretation and use. In: *Proceedings of the IEEE Workshop on Visual Languages*, Dallas, U.S.A., pp. 149–158.
89. J. Bertin (1981) *Graphics and Graphic Information Processing.* Walter de Gruyter and Co., Berlin.
90. N. Bevan & M. Macleod (1993) Usability assessment and measurement. In: *The Management of Software Quality* (M. Kelly, ed.). Ashgate Technical/Gower Press.

91. T. Catarci, S. K. Chang, M. F. Costabile, S. Levialdi & G. Santucci (1996) A graph-based framework for multiparadigmatic visual access to databases. *IEEE Transactions in Knowledge and Data Engineering* 8, 455–475.
92. T. Catarci, S. K. Chang & G. Santucci (1992) A unified framework for providing multi-paradigm visual access to databases. *Proceedings of 2nd International Computer Science Conference*, Hong Kong, pp. 196–202.
93. T. Catarci & M. F. Costabile (eds) (1995) Special issue on visual query systems. *Journal of Visual Languages and Computing* 6, 1–99.
94. T. Catarci & M. F. Costabile (eds) (1996) Special issue on visual query systems. *Journal of Visual Languages and Computing* 7.
95. T. Catarci, M. F. Costabile, S. Levialdi & G. Santucci (eds) (1994) *Proceedings of the International Workshop on Advanced Visual Interfaces AVI'94.* ACM Press, New York.
96. T. Catarci, M. F. Costabile & M. Matera (1995) Visual metaphors for interacting with databases. *ACM SIGCHI Bulletin* 27, 15–17.
97. T. Catarci & I. F. Cruz (1996) Special Issue on information visualization. *SIGMOD-RECORD* 24, pp. 14–54.
98. T. Catarci & G. Santucci (1995) Diagrammatic vs. textual query languages: a comparative experiment. In: *Proceedings of the IFIP W.G. 2·6 Working Conference on Visual Databases* Lausanne, pp. 57–74.
99. T. Catarci, G. Santucci & M. Angelaccio (1993) Fundamental graphical primitives for visual query languages. *Information Systems* 18, 75–98.
100. T. Catarci & L. Tarantino (1995) A hypergraph-based framework for visual interaction with databases. *Journal of Visual Languages and Computing* 6, 135–166.
101. R. G. Cattell (1980) An entity-based database user interface. In: *Proceedings of the ACM SIGMOD Conference Management of Data*, pp. 144–150. Santa Monica, California, May 14–16, 1980. (P. P. Chen, R. Claysprowis, eds). ACM Press, New York.
102. A. K. Chandra (1988) Theory of database queries. In: *Proceedings of the ACM Symposium on Principles of Database Systems*, pp. 1–9. Austin, Texas, March 21–23, 1988. ACM Press, New York.
103. S. K. Chang (1989) *Principles of Pictorial Information Systems Design.* Prentice-Hall, Englewood Cliffs, NJ.
104. S. K. Chang, M. F. Costabile & S. Levialdi (1992) A framework for intelligent visual interface design for database systems. In: *Interfaces to Database Systems*, Glasgow, 1992, *Series Workshop in Computing.* Springer, Berlin, pp. 377–391.
105. S. K. Chang, M. F. Costabile & S. Levialdi (1994) Reality bites—progressive querying and result visualization in logical and VR spaces. In: *Proceedings of the 1994 IEEE Symposium on Visual Languages*, St. Louis, U.S.A., pp. 100–109.
106. S. K. Chang, M. F. Costabile, & M. Vairo (1996) VR queries and their transformation in a progressive querying environment. In: *Electronic Series Workshop in Computing.* Springer, Berlin.
107. P. P. Chen (1976) The entity relationship model toward a unified view of data. *ACM Transactions on Database Systems* 1, 9–36.
108. E. F. Codd (1970) A relational model for large shared data banks. *Communication ACM*, 13, 377–387.
109. E. F. Codd (1972) Relational completeness of database sub-languages. In: *Data Base Systems* (R. Rustin, ed.). Prentice-Hall, Englewood Cliffs, NJ, pp. 65–98.
110. R. Cooper (ed.) (1993) *Interfaces to Database Systems*, Glasgow, 1992, *Series Workshop in Computing.* Springer, London.
111. R. N. Cuff (1980) On casual users. *International Journal of Man–Machine Studies* 12, 163–187.
112. C. J. Date (1987) *An Introduction to Database Systems*—Vol. I. Addison-Wesley, Reading, MA.
113. A. D'atri & L. Tarantino (1989) From browsing to querying. *IEEE Data Engineering Bulletin* 12, 46–53.
114. J. S. Davis (1990) Experimental investigation of the utility of data structure and E-R diagrams in database query. *International Journal of Man–Machine Studies* 32, 449–459.
115. A. Dix, J. Finlay, G. Abowd & R. Beale (1993) *Human-Computer Interaction.* Prentice-Hall International, Englewood Cliffs, NJ.

116. D. K. Doan, N. W. Paton, A. C. Kilgour, & G. Al-Qaimari (1995) Multi-paradigm query interface to an object-oriented database. *Interacting with Computers* 7, 25–47.
117. U. Eco (1975) *A Theory of Semiotics.* Indiana University Press, Indiana.
118. H. Fujii & R. R. Korfhage (1991) Features and a model for icon morphological transformation. In: *Proceedings of the IEEE International Workshop on Visual Languages*, Kobe, Japan, pp. 240–245.
119. K. Furuya, S. Tayama, E. Kutsuwada & K. Matsumura (1987) Approach to standardize icons. In: *Proceedings of the IEEE Workshop on Visual Languages*, Linköping, Sweden, pp. 29–38.
120. N. Goodman (1988) *Languages of Art.* Hackett Publishing Company Inc., Indianapolis, Indiana.
121. E. M. Haber, Y. E. Ioannidis & M. Livny (1995) OPOSSUM: desk-top schema management through customizable visualizations. In: *Proceedings of the XXI VLDB Conference*, pp. 527–538. Zurich, Switzerland, September 11–15. (U. Dayal, P. M. D. Gray, S. Nishio, eds).
122. D. Harel (1988) On visual formalism, *Communications of the ACM* 31, 514–530.
123. T. Schweickert & M. Hemmje (1996) A graphical user interface to the object-oriented database system VODAK on the basis of the generic visualisation toolkit Lyberworld. In: *Electronic Series Workshop in Computing.* Springer, Berlin.
124. M. Jarke & Y. Vassiliou (1985) A framework for choosing a database query language. *ACM Computing Surveys* 17, 313–340.
125. R. John, S. Kuck & A. Lewe (1987) A recursive query language for the universal relation. In: *Proceedings of the 4th International Workshop Statistical Database Management.*
126. J. Kennedy & P. J. Barclay (1996) Interfaces to databases. *Electronic Series Workshop in Computing.* Springer, London, http.//www.springer.co.uk/eWiC/Workshops/IDS3.html.
127. H. J. Kim (1986) Graphical interfaces for database systems: a survey. In: *Proceedings of the Mountain Regional ACM Conference.*
128. P. J. Kolaitis (1991) The expressive power of stratified logic programs. *Information and Computation*, 50–66.
129. F. S. Liu & J. W. Tai (1989) Some principles of icon construction. In: *Visual Databases Systems* (T. L. Kunii, ed.). North-Holland, Amsterdam, pp. 89–104.
130. G. L. Lohse, K. Biolsi, N. Walker & H. H. Rueter (1994) A classification of visual representation. *Communications of the ACM* 37, 36–49.
131. J. D. Mackinlay (1986) Automating the design of graphical presentations of relational information. *ACM Transaction on Graphics,* 5, 110–141.
132. D. Maier, J. D. Ullman & M. Y. Vardi (1984) On the foundations of the universal relation model. *ACM Trans. on Database Systems* 9, 283–308.
133. V. M. Markowitz (1983) ERROL: an entity-relationship role oriented query language. In: *Proceedings of the 3rd Conference on Entity-Relationship Approach.*pp. 329–346. Anaheim, California, U.S.A. (C. G. Davis, S. Jajodia, P. A.-B. Ng, R. T. Yeh, eds). North Holland, Amsterdam.
134. H. M. Markowitz, A. Malhotra & D. P. Pazel (1981) The ER and EAS formalism for system modeling, and the EAS-E language. In: *Entity Relationship Approach to Information Modeling and Analysis* (P. P. Chen, ed.). ER Institute, pp. 29–47.
135. A. Massari & L. Saladini (1996) Virgilio: a VR-based system for database visualization. In: *Proceedings of the International Workshop on Advanced Visual Interfaces AVI'96* (T. Catarci, M. F. Costabile, S. Levialdi & G. Santucci, eds). ACM Press, New York, pp. 263–265.
136. B. Myers, J. Hollan & I. Cruz (eds) (1997) Strategic directions in human computer interaction. *ACM Computing Surveys* 28, in press.
137. A. Motro (1989) A trio of database user interfaces for handling vague retrieval requests. *IEEE Data Engineering Bulletin* 12, 54–63.
138. D. Norman & S. Draper (eds) (1986) *User Centered System Design.* LEA, Hillsdale, NJ.
139. R. Rankin (1990) A taxonomy of graph types. *Information Design Journal* 6, 147–159.
140. P. Reisner (1981) Human factors studies of database query languages: A survey and assessment. *ACM Computing Survey* 13, pp. 13–31.
141. P. Reisner (1988) Query languages. In: *Handbook of Human–Computer Interaction* (M. Helander, ed.). North-Holland, Amsterdam, pp. 257–280.

142. G. G. Robertson, S. K. Card & J. D. Mackinlay (1993) Information visualization using 3D interactive animation. *Communication of the ACM* 36, 57–71.
143. G. Rohr (1988) Graphical user languages for querying information: where to look for criteria?. In: *Proceedings of the IEEE Workshop on Visual Languages*, Pittsburgh, U.S.A., pp. 21–28.
144. L. A. Rowe & K. A. Shoens (1992) A form application development system. In: *Proceedings of the ACM-SIGMOD Conference on the Management of Data.* San Diego, California, June 2–5, pp. 71–80. M. Stonebraker (Ed.), SIGMOD Record 21(2), June 1992.
145. G. Santucci & F. Palmisano (1994) A dynamic form-based data visualiser for semantic query languages. In: *Interfaces to Database Systems* (P. Sawyer, ed.), *Series Workshop in Computing.* Springer, London, pp. 249–265.
146. P. Sawyer (ed.) (1994) *Interfaces to Database Systems*, Lancaster, 1994. *Series Workshop in Computing.* Springer, London.
147. M. E. Senko (1978) FORAL LP: design and implementation. In: *Proceedings of the IV VLDB Conference*, pp. 255–267, West Berlin, Germany, September 13–15, 1978. (S. Bingyao, ed.).
148. D. Shipman (1981) The functional data model and the data language DAPLEX. *ACM Transaction on Database Systems* 6.
149. B. Shneiderman (1982) The future of interactive systems and the emergence of direct manipulation. *Behaviour and Information Technology* 1, 237–256.
150. N. C. Shu (1985) Visual programming languages: a dimensional analysis. In: *Proceedings of the International Symposium on New Directions in Computing*, Norway.
151. S. Sippu & E. Soisalon-Soininen (1988) A generalized transitive closure for relational queries. In: *Proceedings of the International Conference on Principle of Database Systems*, pp. 325–332. Austin, Texas, March 21–23, 1988, ACM Press, NY.
152. E. R. Tufte (1990) *Envisioning Information.* Graphics Press, Cheshire, Connecticut.
153. R. Tufte (1983) *The Visual Display of Quantitative Information.* Graphics Press, Cheshire, Connecticut.
154. M. Y. Yen & R. W. Scamell (1993) A human factors experimental comparison of SQL and QBE. *IEEE Transactions on Software Engineering* 19, 390–402.
155. G. A. Wilson, E. A. Domeshek, E. L. Dascher & J. S. Dean (1983) The multipurpose presentation system. In: *Proceedings of the 9th VLDB Conference*, pp. 56–69. Florence, Italy, October 31–November 2. (M. Schkolnick, C. Thanos, eds). VLDB Endowment/Morgan Kaufmann.
156. G. A. Wilson & C. F. Herot (1980) Semantics vs. graphics—to show or not to show. In: *Proceedings of the ACM SIGMOD Conference on Management of Data*, pp. 183–197.
157. J. E. Ziegler & K. P. Fahnrich (1988) Direct manipulation. In: *Handbook of Human–Computer Interaction* (M. Helander, ed.). North-Holland, Amsterdam, pp. 123–133.
158. M. M. Zloof (1976) Query-by-example: operations on the transitive closure. Technical Report RC5526, IBM Research, Yorktown Heights.